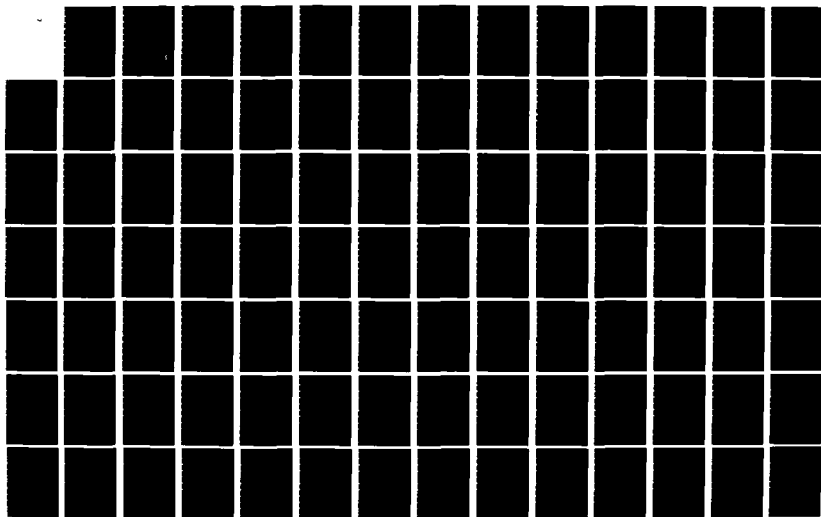
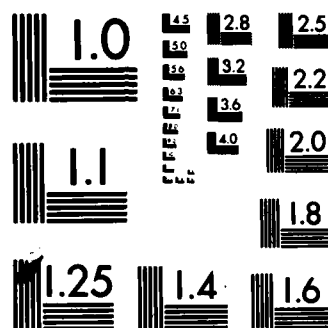


AD-A138 060

A PROPOSED MILITARY PLANNING TASK SIMULATOR USING ROSS 1/2
LANGUAGE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB
OH SCHOOL OF ENGINEERING G H GUNSCH ET AL. DEC 83
AFIT/GE/EE/83D-24 F/G 5/1 NL

UNCLASSIFIED





ADA138060



A Proposed Military Planning Task
Simulator Using the ROSS Language

Thesis
AFIT/GE/EE/83D-24

Gregg H. Gunsch
Capt USAF

Bob V. Hebert
1st Lt USAF

UIC FILE COPY

DTIC
ELECTE
S FEB 21 1984

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

D

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION	1
3	1
Distribution	1

84 02 17 051

AFIT/GE/EE/83D-24

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
AI	



A Proposed Military Planning Task
Simulator Using the ROSS Language

Thesis
AFIT/GE/EE/83D-24

Gregg H. Gunsch
Capt USAF

Bob V. Hebert
1st Lt USAF

DTIC
ELECTE
S FEB 21 1984

D

Approved for public release; distribution unlimited.

A Proposed Military Planning Task
Simulator Using the ROSS Language

Thesis

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Gregg H. Gunsch, B.S.	Bob V. Hebert, B.S.
Capt	1st Lt
USAF	USAF

Graduate Electrical Engineering

December 1983

Approved for public release; distribution unlimited

PREFACE

This thesis is intended to lay the groundwork for the construction of simulators of the military planning process. The techniques used are those of artificial intelligence. The conceptual framework of the simulator is developed, and then an example system is implemented using the ROSS programming language in order to demonstrate certain aspects of the design.

The example system (SAMPLE) performs several of the decision tasks required of an Army Hawk (a low to medium altitude surface-to-air missile system) battalion commander when choosing emplacement sites for his firing batteries. The decision processes implemented in the example are only an approximation of those used by the commander, and as such are somewhat oversimplified. The intent of the example is to demonstrate the methodology of transferring the real-world expertise of the commander into a computer simulation, not to provide an accurate simulation of the emplacement task. We are not Hawk commanders: we do not have the expertise.

We would like to take this opportunity to thank those who helped us the most: our wives -- Cherry and Laura. Without their loving support and tolerance, this thesis

would not have been possible. We would also like to thank our children; who have gone so long without their fathers. We hope they remember who we are.

Special thanks to Captain Rob Milne, our advisor, for the encouragement to break new ground, and to Doctor Matthew Kabrisky for his wise guidance and gentle criticism. Finally, a warm "thank-you" goes to our librarians; especially Linda Stoddart for always finding the references we needed. We don't know what we would have done without her.

Gregg H. Gunsch

Bob V. Hebert

Table of Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	vii
Abstract	viii
I. Introduction	I - 1
Problem Definition	- 1
Scope	- 2
Assumptions	- 7
II. Background	II - 1
Planning	- 1
Summary of Current Knowledge	- 6
Our Philosophy	-13
III. Planning Task Simulator	III - 1
Introduction	- 1
ROSS - An Object-Oriented Simulation Language	- 2
Extending the ROSS Environment	- 5
IV. Planning Task Simulator Design Procedure..	IV - 1
Introduction	- 1
General Design	- 4
Detailed Design	- 7
V. SAMPLE General Design	V - 1
Introduction	- 1
Task Description	- 2
Environment Description	- 7
Knowledge Domain Description	-11
User Behavior Specification	-16

Table of Contents

	Page
VI. SAMPLE Detailed Design	VI - 1
Simulator Design Refinement	- 1
Context-Level Micro-Expert Design	-11
Micro-Expert Design	-12
VII. Summary	VII - 1
Introduction	- 1
Task Algorithm Results.....	- 1
Conclusions and Recommendations	- 4
Bibliography	BIB - 1
Appendix A: Planning Tactics	A - 1
Appendix B: Site Selection Objectives	B - 1
Appendix C: SAMPLE User's Guide	C - 1
Appendix D: Planning Task Simulator Designer's Guide	D - 1
Appendix E: Using Terrain in a Planning Task Simulator	E - 1

List of Figures

Figure	Page
IV-1: Simulator Structure	IV - 2
V-1: Typical Deployment Scenario	V - 4
V-2: Task Inputs and Outputs for the Site Selection Task	V - 5
V-3: Organizational Structure of a Triad Battalion in Direct Support of a Division	V - 8
V-4: SAMPLE Communications Model	V -10
VI-1: SAMPLE Functional Modules	VI - 2
VI-2: SAMPLE Top-level Hierarchy	VI - 3
VI-3: Leading and Supporting Actor Hierarchies.	VI - 4
VI-4: Logical Actor Structure	VI - 7
VI-5: Micro-Expert Component Interaction	VI -13
B-1: Mutual Support and Overlapping Fire	B - 3
B-2: Typical Deployment Scenario	B - 5
C-1: UNIX Directory Structure	C -10

List of Tables

Table	Page
VII-1: Scenario Definition	VII - 2
VII-2: Candidate Site	VII - 2
VII-3: Site Scoring	VII - 3
VII-4: Pair Scoring	VII - 3
C-1: Scenario Definition Components	C - 8

ABSTRACT

Planning, particularly military planning, is viewed as the integration of subplans, each of which serves to accomplish a subgoal of the original goal. This model serves as the conceptual framework of the planning task simulator. The simulator is an environment in which task specialists, modeled by expert systems, are used to formulate courses of action to meet the given subgoals.

Portions of the simulator have been implemented as a demonstration of the feasibility of this approach, as well as a guide for future developers. The language used was ROSS, an object-oriented simulation language embedded in Franz Lisp or MacLisp and developed by the Rand Corporation. The implementation was done on the Air Force Institute of Technology's Scientific Support Computer, a VAX 11/780. The dialect used was the Franz Lisp version of ROSS.

Several system functions such as data storage and retrieval mechanisms, a communications channel model, design aids, and debugging aids were implemented to provide a workbench for the development of an extended

simulator. One task specialist was also constructed: a small-scale expert system that performs the planning involved in selecting emplacement sites for Hawk medium-range surface-to-air missile (SAM) batteries in direct support of an Army division in the defensive posture.

Although the implemented task specialist is highly constrained and only an approximation of the real-world counterpart, it serves to demonstrate the ease with which task specialists can be constructed. The transparency, modularity, and power of ROSS, with the additional system functions, provides a robust environment in which to simulate real-world planning algorithms.

CHAPTER ONE

INTRODUCTION

1.1. PROBLEM DEFINITION

Both Eastern and Western analysts agree that success on the modern battlefield relies on effective military planning. Because of the increasing complexity of expected battlefield scenarios and the range of specialized knowledge required of all commanders, effective planning is becoming increasingly difficult. Three areas of current research are potentially capable of aiding battlefield commanders.

First, several Air Force programs are currently developing automated decision aids for military planning tasks. Typically, such aids serve a data reduction function. Second, several researchers have developed models of the planning process. Study of these models has led to the study of planning heuristics designed to improve the effectiveness of human planners. Third, automated systems exist that simulate the problem-solving processes of human experts within a limited domain; such automated systems are known as "expert systems." We believe that the integration of these three research areas could lead to the development of a planning task

simulator, in which military task specialists, modeled by expert systems, can be used to formulate courses of action aimed at achieving assigned military objectives.

Such a simulator would be an effective decision aid for the battlefield commander. Not only would it serve the data-reduction function of current decision aids, but would also embody much of the expertise of the commander's specialization. The simulator would provide the most reasonable alternatives to the commander, who could then focus his talents on only promising solutions. Additionally, the explanation capabilities possible in expert systems could provide the commander with explicit illustrations of the strong and weak points of the alternatives, and the reasoning used in selecting one over the others.

The intent of this thesis is to demonstrate one possible approach to the implementation of such a planning task simulator. To this end, we have defined the following three thesis objectives:

1. Develop a set of techniques and representation schemes appropriate for a planning task simulator.
2. Develop a design procedure for implementing a planning task simulator.
3. Demonstrate the procedure, techniques, and representation schemes by designing and implementing an example task simulator that performs the site selection task for a low-to-medium-altitude, radar-guided surface-to-air missile (SAM) system.

Chapter Three will define the programming techniques and representations necessary to implement the planning task simulator. Chapter Four presents the design procedure used to construct the simulator. The demonstration system's general design is presented in Chapter Five, and Chapter Six gives the detailed design.

1.2. SCOPE

The design methodology incorporates the techniques of artificial intelligence. According to Patrick Winston, "Artificial Intelligence is the study of ideas which enable computers to do the things that make humans seem intelligent" (Winston, 1979: 1). A complete system, given a well-defined doctrinal procedure for planning a specific military task, will be capable of automatically predicting a course of action indistinguishable from that which a human planner would arrive at if he used the same procedure.

The system design concepts are illustrated by the design and implementation of a small-scale expert system which models some of the planning activities involved in emplacing Hawk surface-to-air missile systems. By "small-scale" we mean that only those elements of the planning process which are necessary to solve the problem

are implemented; several features normally associated with expert systems are not implemented. The expert system is to serve as a guide: an example of modeling military planning activities. Our primary goals in developing this example system are to make it simple enough to clearly illustrate our design philosophy, and yet complex enough to demonstrate that real-world problems can be solved by such a system. We hope that the techniques and design tools developed in this thesis will aid future researchers in the development of planning simulators for other military task domains.

All automated systems require some sort of interface between the computer program and the user. Expert systems generally incorporate a complex input/output package, typically including computer-generated graphics, natural language communication with the user, or both. Such systems also incorporate a query mechanism which allows the user to trace the reasoning steps followed in generating a given output. The query mechanism is sometimes quite complicated, allowing interactive editing and querying of the system knowledge base. Implementation of such interfaces is beyond the scope of this effort. Typical expert systems that include these interfaces require approximately five man-years to design and implement (Davis, 1982: 10). Instead, a simple input/output package, based on a menu-driven input format

and tabular output has been implemented. A simple trace mechanism is also included to perform the query function.

A complete planning simulator would consist of a number of task specialists operating within an overall planning model. A complete implementation of the overall model is not attempted; only those modules required to connect a single task specialist (the example expert system) to the user are implemented. The components of the simulator that integrate the outputs of several task specialists have not been implemented.

We will approach the use of terrain data by requesting information from a map-reader module, using symbolic descriptions of the information desired. For this effort, the map-reader will ask the user for the information being sought, and a human map-reader will have to abstract the data and determine the reply. However, the region-finding algorithms and methods of determining connectivity that are required to abstract a digital map into useful terrain representations are available and discussed in the Handbook of Artificial Intelligence (Fiegenbaum, 1981). We are convinced of the implementability of the terrain-data-base to terrain-abstraction conversion and automation of the map-reader function, provided the mass-storage and related computational details can be resolved.

As stated in the problem definition, this thesis demonstrates one possible approach to the implementation of a planning task simulator. The thesis accomplishes the following three objectives:

1. Development of a set of techniques and representation schemes appropriate for a planning task simulator.
2. Development of a design procedure for implementing a planning task simulator.
3. Demonstration of the procedure, techniques, and representation schemes by the design and implementation of an example task simulator.

The example system is constructed to operate in a manner similar to the planning procedures of the human; that is, the steps that the system takes in working through the problem are English-like and straight-forward (as opposed to reams of magical code such as an operations research project). Additionally, the task algorithm we have used is intentionally simplistic and tightly constrained. This was done to present a clear demonstration of our design procedure and the use of an assortment of techniques and representations, without focusing attention on the details of the task itself. We hope that this approach will provide future simulator developers with a rich example to follow in the implementation of their systems.

1.3. ASSUMPTIONS

The following assumptions are incorporated into the solution of the problem:

1. Within limited domains, human problem-solving activity may be successfully modeled by a computer program. This is the fundamental assumption underlying the current research in expert systems. The demonstrated performance of several available expert systems tends to support the validity of this assumption (Feigenbaum, 1981: Vol 2). The term "limited domains" refers to the present inability of computer scientists to develop an automated problem solver that can operate in more than one field of expertise. This inability limits the scope of any project based on current expert systems. For example, a military planning simulator can not be extended into a economic forecaster without a major rework.
2. Military planning involves the coordination of and compromise among several independent task specialists. This assumption is the basis of the Hayes-Roth planning model: that planning can be accomplished through the coordinated efforts of specialists, each working on a particular part of the problem (Hayes-Roth, 1980). The observed

specialization of current planning staffs justifies the inclusion of this assumption. Adoption of this model requires the future development of a mechanism to implement the coordination and compromise function.

3. Military planning is significantly constrained by national policy, established doctrine, and accepted practices. These constraints are what make military planning activities a "limited" domain and thus facilitate the computer simulation of the activity.

CHAPTER TWO

BACKGROUND

2.1. PLANNING

2.1.1. Definition of Planning

Planning is defined as the formulation of an intended course of action aimed at achieving a goal (Hayes-Roth, 1980). This definition implies that planning may be decomposed into three components: the goal, the course of action, and the formulation of the specific steps of the course of action.

The goal is simply a desired result that the planner is trying to achieve. For all except the simplest of goals, the course of action will likely consist of a sequence or set of steps which must be performed to reach the goal. Each step may then in turn be viewed as the goal of a simpler "subplan". The integration of these subplans into a coherent effort to reach the ultimate goal then completes the planning process.

This is not meant to imply that every goal can be factored into a set of subgoals. However, pursuit of a non-factorable goal of any complexity will quickly become

an unmanagable task for a human planner if he is expected to produce a solution that is correspondingly complex and non-algorithmic (not step-by-step in nature). Some planning tasks are solved through flashes of insight, but the majority are solved by "divide and conquer" techniques: abstraction, bugging, and psuedo-reduction. Appendix A contains an explanation of these techniques as applied to human planning. Even if the goal is not cleanly factorable into subgoals, there are still certain items of interest that are dominant. The dominant items, whether drivers or constraints, should be used to shape the solution, and the lesser items to refine the details. There may be conflicts, but these are introduced in the course of making a first-cut approximation to a solution. The techniques for resolving certain classes of these conflicts and the methods of selecting an approach to the problem are classified as tactics of "meta-planning". Meta-planning is discussed in Secton 2.2.1. and in Appendix A.

Study of the human planner reveals that the good (robust) planner has at his disposal a variety of techniques with which to approach the task (Sacerdoti, 1979). He is opportunistic (Hayes-Roth, 1980) and chooses the technique to fit the problem. The good planner has, as well as develops, heuristics to guide the selection of the techniques. A computer program, intended to be a

robust planner, will also require these properties. It must have the capability to recognize the subgoals and evaluate the importance of interesting data. It will then decide which technique or techniques to apply and begin the search for a solution. The program or planner must also develop metrics with which to evaluate the course of the search. To effectively plan and evaluate a course of action, even a rough sketch of one, the planner must have a large knowledge base extending the breadth of the planning domain, with sufficient detailed knowledge to satisfactorily assure itself of the feasibility of the implementation.

Many planning tasks require a great deal of specialized knowledge in order to form a solution. In the case of the human, such tasks are delegated to more specialized planners. The manager or executive only needs to know that the solution will be found, and who can find it. He delegates the task, and is then free to work in his proper domain, which is a broader, but shallower piece of the "big picture". The person who received the delegated task presumably is an expert in the specialization, and we shall refer to him as a "task specialist".

As the tasks become increasingly narrow and specialized, the techniques that can be applied also

become less general. Often, the specialist uses the same procedure with nearly every task assigned him, since the tasks a specialist receives are usually similar. These often-used, cut-and-dried procedures are the kinds currently being implemented with expert systems. In this light, an expert system is simply an implementation of the abilities of the task specialist, within a very limited problem domain, and with a limited knowledge base.

The manager, on the other hand, is given tasks that are not as well-defined and require broader knowledge. He knows what resources and experts are available, and integrates their results into a course of action using the techniques previously mentioned. For the most part though, managers are middle managers. That is, they delegate work to those they manage, but at the same time receive their tasks from higher authority. From the viewpoint of his superior, the middle manager is a specialist: specializing in planning a portion of the subtasks that the superior can generate when formulating his own plans. This hierarchical structure can extend indefinitely, with each person being viewed by his subordinates as a more general planner, and by his superiors as a specialist in a narrower planning domain. (In this instance, the terms subordinate and superior refer to the chain of delegation of components of the problem.)

2.1.2. Military Decision-Making as a Planning Function

Military commanders at all levels are required to produce decisions which result in the accomplishment of military objectives. These commanders may have subordinate units that provide the input data or develop possible courses of action needed to formulate the overall plan. The task of the commander then, is to convert the input data and list of alternatives into a decision on the best approach to meeting his assigned objectives.

Military decision-making, as just defined, is therefore a planning function in the general sense. Yet, several characteristics of military decision-making serve to distinguish it from other forms of planning:

1. The military commander is responsible for developing plans to meet goals often generated by higher authority or the actions of the enemy (and not by the commander himself).
2. Military plans are often required under stringent timing constraints. Additionally, commanders frequently do not have the luxury of not making some decision.
3. The possible courses of action that a commander may pursue are often significantly restricted by national policy, established practices, and military doctrine.
4. While a commander may task subordinate units to help produce a decision, the commander remains ultimately responsible for his decision. Therefore, two types of expertise are generally required of all commanders:
 - a. Expertise in an assigned specialty area (naval destroyer captain or an army tank commander, for example).
 - b. Expertise in general resource management and assessment of alternatives generated by subordinates.

The military planning environment therefore consists of the integration and coordination of decision-making activities conducted at all command levels. Each unit (person or group of persons) is required to accomplish its assigned mission, and can in turn assign its subordinate units certain portions of the task. Units also trade information with other units not in the chain of command (lateral communications). Military doctrine and delegation of authority ensure that each unit has a well-defined set of behaviors relating to received tasking, generated orders, and lateral communications.

2.2. SUMMARY OF CURRENT KNOWLEDGE

2.2.1. General Planning

As previously stated, a plan is the course of action formulated to achieve a goal. The most obvious property of a plan is that a plan is the integration of a collection of subplans, each of which serves to achieve a subgoal. When all subgoals are accomplished and conflicts among them resolved, the original goal will have been achieved.

Each subplan is in itself a plan, and so can be further subdivided into smaller subplans. Eventually, the plan will decompose into a collection or sequence of simple and readily executable actions. Formulating this collection into a workable course of action requires ordering, combining, and compromising between the simple actions. Compromise is necessary because, quite often, actions aimed at achieving one subgoal conflict with the achievement of another. A general planner, one that does not have a predefined task to accomplish, must have at its disposal a set of tactics for resolving these conflicts (or avoiding the generation of conflicts altogether). The paper "Problem Solving Tactics" (Sacerdoti, 1979) discusses a handful of approaches to producing workable plans. These tactics are discussed in Appendix A.

This general problem-solving procedure of decomposing the problem into subparts and then integrating the subplans is powerful, in that it is applicable to a wide variety of real-world situations. However, once a subgoal has been adequately defined, it will probably be advantageous to use a task-specialist to generate the subplan. A specialist has a great deal of knowledge relating to the particular subgoal; this expertise includes heuristics -- time-tested methods of attacking the particular problem. These heuristics serve to trim a general-purpose planner into one that is tuned to the

particular specialty, capable of producing good solutions with less search (therefore, less computation) than the general-purpose planner.

Specialists are used extensively in the "opportunistic planning model" developed by Barbara Hayes-Roth, et al. This model views planning as the cooperative efforts of many independent plan specialists. Each specialist makes tentative decisions about the plan from the data it has on hand. The decision is posted on a data structure common to all the specialists: the blackboard. Through cooperation and transfer of knowledge, a course of action is derived. This project focuses on problems analogous to the naval tactical planning problem: How should the decision-maker move ships from their current locations to particular task-force objectives? (Hayes-Roth, 1980).

The Hayes-Roth approach uses top-down and bottom-up planning specialists. The top-down specialist generates the subgoals from the given goal and directs the search for the solution. Bottom-up specialists provide the higher-level planners with data about execution details which constrain the direction of the search. The term "opportunistic" refers to the fact that there is no strong predefined organization of planning activity. The executive control process, which invokes and schedules the

specialists is also opportunistic. This approach allows the planning system to make use of all available information to guide the search for the solution.

An alternative to the opportunistic planning model is offered by Mark Stefik in his paper "Planning and Meta-Planning" (Stefik, 1981). While the fundamental concepts of the two approaches are very similar, the implementations are different. The method put forth by Stefik involves the use of a meta-planner to control the activity of the other planning stages. A meta-planner is a planner that determines how to formulate a plan. Stefik's planning structure consists of planners whose control and interactions are handled by meta-planners, with meta-meta-planners (so to speak) controlling the meta-planners, and so on. This is part of a line of research aimed at enhancing the power of a problem solver by allowing it to reason about its own reasoning process.

A comprehensive look at planning and meta-planning is found in the book Planning and Understanding by Robert Wilensky. According to Wilensky, meta-planning addresses "the problems of goal interactions, and plan debugging and modification" (Wilensky, 1983: 23). This level of planning uses knowledge about the planning process itself, in terms of meta-goals, meta-plans, and meta-themes. Meta-goals are the goals of the process of planning. For

example, "resolve conflicts" and "find a workable solution" are meta-goals of the planning process. Meta-plans are those plans and techniques that can be applied to achieve the meta-goals. Meta-themes are the situations that group the meta-plan and meta-goal knowledge. The four meta-themes encountered by Wilensky are:

1. Don't waste resources.
2. Achieve as many goals as possible.
3. Maximize the value of the goals achieved.
4. Avoid impossible goals.

Meta-planning is accomplished through the use of specialists. Wilensky's meta-planning specialists are called "meta-entities". Each meta-entity contributes toward meeting a particular meta-theme. Using the meta-themes as guidance, the complete planning structure formulates the problem in terms of its subgoals, and then solves the problem by solving for the subgoals and integrating the partial solutions. Appendix A contains more on planning and meta-entities.

2.2.2. Military Planning

In the middle ground between general-purpose planners and special-purpose subplanners are planners that embody portions of both. The military officer is one such planner. He must have detailed knowledge relating to his

assigned duties. In other words, he must be an expert at his job. Additionally, he must have knowledge of how he fits into the "big picture" and enough general planning ability to formulate his plan to fit in better with his superior's other subplans. Finally, he must have enough knowledge of his subordinates' duties and abilities to avoid assigning tasks that they cannot accomplish, and the general planning abilities to integrate the subplans from his subordinates. These points illustrate the fact that a plan by one military planner can be a subplan to his superior, and that a planner may delegate the development of subplans to his subordinates.

Since the simulator is intended to be generally applicable to military planning, we must be able to structurally model both Western and Eastern planning activities. The book by Druzhinn (Druzhinn, 1972) is a Soviet view of the military decision-making process. The purpose of Concept, Algorithm, Decision, as stated in the first chapter, is to contribute to the development of the theory and technique of decision-making. Emphasis is placed on the role of automation in the control and management of military operations and of the intrinsic capabilities of computer technology when applied to military decision-making. The author develops a general model of decision-making which is consistent with our simulator design.

Another viewpoint of the Soviet decision-making process is put forward in the paper "Modeling Soviet Defense Decisionmaking" (Alexander, 1980). Alexander describes the decision-making environment in terms of "actors". High-level actors have the authority to make decisions and produce policy. They often face problems of conflicting goals that require political action to resolve. The low-level actors implement the actions required by higher-level actors, and generate information. They often face problems that require high-level solution and put forward proposals, initiatives, and alternatives. They generate conflict among themselves that must be resolved by the political decisions of higher-level actors.

The "behaviors" of high-level actors are dominated by the influences of politics, personalities, values, and national goals. Low-level actor behaviors are tightly constrained by doctrine, bureaucratic processes, and the decisions of the high-level actors. This is not to say that low-level actors do not influence the decisions of the high-level actors; on the contrary, they do influence the decisions of the higher-level decision-makers through the information, proposals, and alternatives they provide. While this model is intended for the Soviet decision-making process, it can be readily extended to other decision-making environments (for example, U.S.)

with a reordering of priorities and changes in the effects of the influences.

2.3. OUR PHILOSOPHY

We hold that the same planning tactics being taught humans can be implemented as behaviors of a generalized planning system. Codifying the abilities to resolve conflicts, evaluate interactions, abstract situations and data, and intelligently change tactics mid-stream could easily be dismissed as an impossible task: the mechanisms used by the human planner are themselves poorly understood. How could one expect to program a machine to do it? However, progress is being made in defining the actions taken by robust human planners and problem-solvers, and as more is understood, more can be incorporated into the simulator. In fact, judicious use of the simulator can direct planning research since the computer demands definition of much knowledge taken for granted (common sense). In picking apart common-sense assumptions, one may find that the assumptions were in fact only partially true, like any good heuristic. One of the distinguishing features between good and poor planners is the ability to recognize when the situation has components that conflict with the common sense assumptions. Poor planners don't question their

assumptions, and may not even know why the assumption exists. Therefore, a thorough understanding of all aspects of the problem domain, including the knowledge taken for granted, is necessary for a good planner.

We have acknowledged the fact that building a good planning system is very difficult. This thesis is intended to take a step in that direction and ease some of the endeavor by providing an extensible, evolutionary system that can grow as new tasks are defined. As planning tactics are refined, they can be incorporated in the higher-level generic planning procedures in a manner like Wilensky's meta-entities. As new task specialists are defined, they can be plugged into the system as the low-level specialized planners.

We do not make the claim that this type of planning system is capable of finding solutions to all planning tasks, just those solvable by humans. As stated earlier, this approach requires that complex goals be factored into simpler subgoals: a "must" for the limited resources of the human mind. To those who pose the question, "What about the goal that cannot be factored into subgoals?", we respond, "How would you achieve it?" In the answer lies the expertise. It is our belief that even that goal, upon examination, has components that can be treated separately -- in a subgoal-like manner.

CHAPTER THREE
PLANNING TASK SIMULATOR
DESIGN TECHNIQUES

3.1. INTRODUCTION

This chapter is intended to define the programming techniques, communications model, and the knowledge representation schemes necessary to implement a military planning task simulator. Chapter Four presents a design procedure used to construct the simulator, while Chapters Five and Six describe how this procedure was followed in the design and implementation of an example simulator.

Because many of the concepts presented in this chapter are more easily understood in the context of an example, the discussion is purposefully brief; additional information is presented in Chapters Five and Six, as well as Appendix E.

3.2. ROSS -- AN OBJECT-ORIENTED SIMULATION LANGUAGE

3.2.1. The Need for an Object-Oriented Language

Most, if not all, of the decisions made by a tactical battlefield commander are "data-driven," in that the possible alternatives the commander has to choose from are highly constrained by the current battlefield situation, available terrain, and reconnaissance information. There is a correspondence between the data-driven nature of planning tasks and the "trigger (message)/response (behavior)" structure of object-oriented languages that make them inherently well-suited for simulating such tasks. Additionally, representing procedural knowledge (task algorithms, e.g.) is extremely straightforward in an object-oriented language. While production systems are adequate for representing this procedural knowledge, an object-oriented language can well represent both the procedural knowledge (responses) and the declarative knowledge (data). Therefore, an object-oriented language is the natural choice for implementing a planning simulator.

3.2.2. ROSS

We have chosen ROSS as the implementation programming language: an object-oriented language developed at the RAND Corporation for the purpose of constructing simulations. The language had its conceptual origins in SMALLTALK and DIRECTOR and is embedded in either MacLisp or Franz Lisp (McArthur, 1982). It contains primitives for symbolic pattern matching and provides for the inheritance of both behaviors (responses to inputs -- procedural knowledge) and attributes (data -- declarative knowledge). The object-oriented structure, near-English programming syntax, extensibility, and ease of designing with the language are the primary reasons we chose ROSS.

The principle programming construct in ROSS is the "actor", which is a set of Lisp functions (in our implementation, Franz Lisp) or ROSS commands that are invoked when an incoming "message" matches a predefined pattern. Actors may send messages to themselves or to other actors. The program thus consists of a set of actors passing messages back and forth. Useful work is accomplished when the proper message triggers appropriate responses from the proper actor.

Actors are created in a hierarchy to take advantage of the inheritance mechanisms. An actor will inherit the

behaviors and attributes of the actor that created it (its parent). The parent inherits from its parent, and so on; therefore, an actor will inherit from all of its ancestors. Multiple parents are allowed, thereby providing the ability to construct tangled hierarchies of any desired complexity.

The branch nodes of this hierarchy usually consist of generic actors. The leaves are usually instance actors. (We say "usually" because the trees can be constructed in any fashion.) A generic actor consists of behaviors that are common to all actors of a class; for instance, a generic "person" actor should contain behaviors for walking, eating, talking, etc. since these behaviors are common to all people. The generic actor also contains the common attributes of the members of the class. These can be thought of as default values. An instance actor represents an individual or functionally distinct element. It may contain behaviors and attributes that set it apart from the other members of the class, or it may inherit all of its characteristics.

We have chosen to place all behaviors in generic actors, and only the "customized" attributes in the instance actors. This method localizes the storage of the information as high up in the hierarchy as possible. This allows the largest number of actors to inherit the

information. Also, with the common knowledge moved away from the less general actors, the focus of attention of the simulator designer can be on the problem-domain-specific aspects of the task.

The initial ROSS environment contains several predefined actors and behaviors for the creation, removal, and modification of new actors. Additionally, behaviors for manipulating the behaviors and attributes of actors are predefined, as are a number of other behaviors. For a complete description of these behaviors, the reader is directed to the ROSS manual (McArthur, 1982).

3.3. EXTENDING THE ROSS ENVIRONMENT

While the initial ROSS environment is very powerful, we have developed the following actors to aid in the design of a planning task simulator:

ACTOR	FUNCTION
1. Someone	Planner Model
2. Leading-Actor	Planner Model
3. Link	Communication Model
4. Network	Communication Model
5. Bulletin-Board	Data Storage and Retrieval
6. Personal-Memory	Data Storage and Retrieval
7. Scratchpad	Data Storage and Retrieval
8. User-Interface	Design Aid
9. Designer	Design Aid
10. Comm-Log	Debugging Aid
11. Historian	Debugging Aid

Collectively, these actors are known as "system" actors, to distinguish them from task-specific actors that differ from one simulator to the next. In addition to the system actors, we have found it convenient to conceptually distinguish between two types of task-specific actors: those actors with real-world analogues (henceforth called "leading actors"), and those actors without analogues (henceforth called "supporting actors"). Supporting actors are used to implement functions required because we are simulating human planning functions with a computer program; for example, the "controller", responsible for the proper sequencing of the simulator, has no real-world counterpart, and thus would be a supporting actor.

3.3.1. Planner Model Actors

The "someone" and "leading-actor" actors contain behaviors that implement general-purpose behaviors of those actors representing real-world planners (leading actors). For example, all leading actors must "know" how to communicate with other leading actors. This knowledge is represented by a leading-actor behavior. Leading actors also have other actors available to them as personal resources, such as the "personal-memory", "scratchpad", and "comm-log" actors. The leading actors we have implemented contain only those behaviors necessary

to perform the demonstration. The behaviors needed to perform general-purpose planning (see Appendix A) have not been implemented.

3.3.2. Communication Model Actors

Message-passing in ROSS is normally unrestricted, in that any actor may send a message to any other actor. To better model the real-world, we have chosen to restrict message-passing: a leading actor can communicate with other leading actors only through the "network" and "link" actors. Collections of these actors correspond to the communication channels available to the leading actor's real-world counterpart, and are defined as follows:

1. Links: provide a communications channel between two leading actors.
2. Networks: provide a communications channel among several leading actors.

Links and networks are referred to collectively as comm nodes, and were introduced for the following reasons:

1. Nodes correspond to real-world communications links, thus enhancing the "simulation" ability of the system. This should make the system easier to understand and debug.
2. Nodes restrict the message traffic within the system, thus making the design more structured and reliable.
3. Nodes provide for a built-in extensibility to command, control, and communications simulation capability, including the capability to add the effects of failures, redundancies, saturation, and jamming.

3.3.3. Data Storage/Retrieval Actors

The "personal-memory", "scratchpad", and "bulletin-board" actors are the primary vehicles for storing declarative data within the simulator environment. The personal-memory actor is designed to store time-changing data accessible to a single leading actor. The scratchpad actor is designed to store time-changing data related to the calculations involved in planning a task, and is especially suited for tabular data. The bulletin-board actor stores time-changing data accessible to several actors. (Data that do not change with time are "hard-wired" into the simulator as actor attributes.) Additionally, the personal-memory actors are capable of a limited amount of inferencing, thereby allowing a less restricted storage or retrieval syntax. For example, if a leading actor is told that the group consists of his superior and his peers, is told that his superior is commander-1, and is also told that the comm-channel to the group is the radio, then the message "fetch the comm-channel to commander-1" would return "radio". The proper result is returned even though the actor was never explicitly told the comm-channel to commander-1.

3.3.4. Design Aids

We have developed two actors whose sole functions are to simplify the simulator design phase. The "user-interface" actor, when given a symbolic message name, will print the associated input prompt on the terminal, read the response, validate the response against a predefined set of constraints, and set the proper system parameter to the input value. If the input is invalid, a diagnostic error message is printed and the user is asked to reenter the value. The "designer" actor allows for the dynamic creation of actor instances, as well as the creation of communications links and networks.

3.3.5. Debugging Aids

The "comm-log" and "historian" actors record message traffic among leading actors. Each leading actor owns a comm-log that records all messages sent to the actor. The historian contains the sum of all messages recorded by comm-logs. These recording actors not only simplify the debugging of the simulator design, but also provide the user with a record of the message traffic during a session.

3.3.6. Representing Constraints

Much of the information that drives the commander's search for a solution is in the form of constraints: mission constraints, environmental constraints, constraints imposed by the enemy, etc. One characteristic of a constraint is that it is desirable or sometimes mandatory to meet the constraint, and undesirable or detrimental not to meet it. Meeting constraints is a meta-goal of the planning process.

Various methods can be used to represent constraints, two of which are the most common. The first is to describe the constraint in the form of a continuous cost function. For example, the constraint of not getting too close to a burning fire can be described by the distance verses radiant energy relation: the amount of radiant energy impinging on a surface is inversely proportional to the square of the distance from the fire. To avoid being singed, one must remain a certain distance from the fire; that distance is determined through the use of the cost function.

The second method is to represent the constraint with a limited number of possible cases (or classes). These cases sample the range of possible values the constraint can take on. In the previous example, the cases

describing the amount of incident heat as the distance increases may use terms like: intolerable, dangerous, uncomfortable, comfortable, noticable, and insignificant. This representation has proven useful in guiding decision heuristics (for example, "If dangerous, move!") and is the type used in the demonstration system. We have found it to be psychologically appealing since the human planner does not evaluate his alternatives at every possible value of the (continuous) cost function, but approximates the constraint to limit the number of computations he must perform. This representation is similar in many ways to "fuzzy set" theories.

The selection of the proper thresholds (the borders between classes) is critical to the effectiveness of the constraint when used to guide heuristics. Suitable metrics must be devised to determine where the thresholds lie. The situation or doctrine may provide many of these metrics, as well as user/simulator interactions ("tinkering" with the system). One example would be the distance from the enemy that a commander could place his resources and still remain reasonably sure of their safety. If the range of ground artillery and mortars is 8 kilometers, then 8 kilometers is a suitable threshold. Placing his resources closer to the enemy than 8km would put them in danger; beyond 8km, they would be safe. Near the threshold, the commander must make a judgement and

other factors would come into play. If he was conservative, being near the 8km threshold would be considered dangerous. If aggressive, then the risk would be accepted. This illustrates the point that some constraints dominate over others, and solutions are found through their interactions.

3.3.7. Modeling Terrain Within a ROSS Environment

All military planners require terrain data to formulate decisions. Currently available sources for the real-world commander include the Defense Mapping Agency's Digital Terrain Map system, and the US Army's 1:50,000 and 1:250,000 scale series maps.

We have developed a specification for a General Military Map Representation (GMMR) that, ultimately, would contain all terrain information that commanders could glean from standard military maps. The GMMR should contain information about elevation, grid locations, terrain cover and foilage, locations of roads, rivers, cities, railroads, and items of specific military interest (see Appendix E). Each terrain-using task specialist would have a map reader adapted to the specific job of extracting and compiling the terrain information that the task specialist will need.

CHAPTER FOUR
PLANNING TASK SIMULATOR
DESIGN PROCEDURE

4.1. INTRODUCTION

The purpose of this chapter is to present a design procedure which can be followed to construct a planning task simulator. Chapters Five and Six describe an example simulator designed according to this procedure.

Figure IV-1 describes the general structure of a planning task simulator. The structure consists of three distinct levels and the interfaces between the layers. The simulator is designed to operate interactively. The user layer contains the functions that the user will perform during the simulation. The inclusion of this layer in the design structure allows the retention of a human "in the loop." The simulator layer contains the system actors and representation mechanisms defined in Chapter Three. This layer effectively buffers the user from the operation of the micro-expert. The task layer contains the micro-expert(s) that actually perform the assigned tasks.

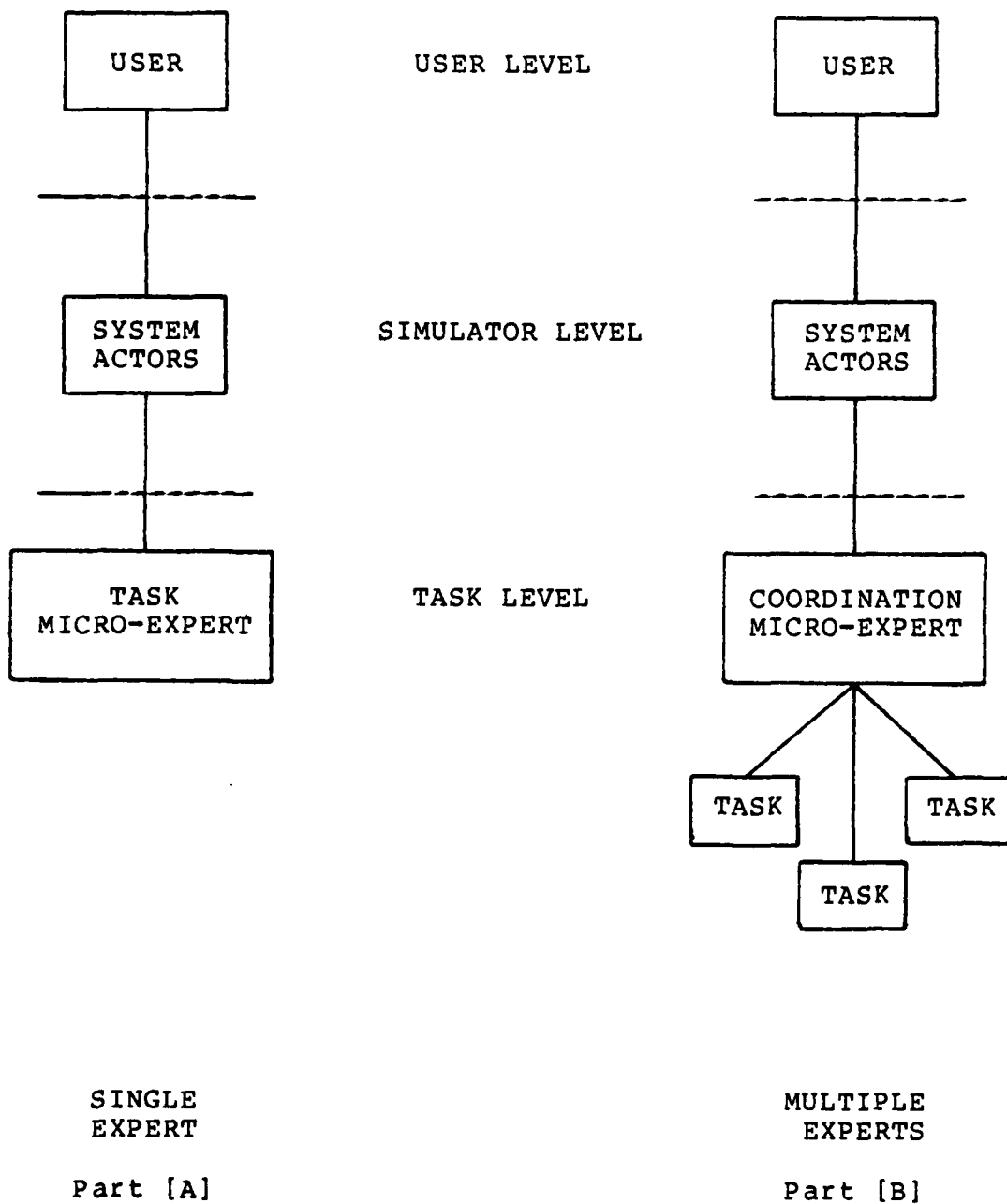


Figure IV-1. Simulator Structure

The figure depicts two different organizations of the task layer. Part [A] shows a single micro-expert interfaced directly to the simulation layer. This structure is appropriate for relatively simple decision-making tasks where a single individual is wholly responsible for developing the possible courses of action. Part [B] shows multiple micro-experts being controlled and coordinated by a special coordinating micro-expert. This structure is appropriate for more complicated planning tasks where several task specialists provide possible alternatives to a higher echelon. The higher echelon planner then evaluates the alternatives and produces the decision. In both cases the simulator layer communicates directly with a single micro-expert, thus simplifying the design. The second case requires the development of a micro-expert whose domain of expertise is the coordination and evaluation of other experts' plans. This micro-expert should contain behaviors like those discussed in Appendix A. The example simulator presented in this thesis demonstrates the structure depicted in Part [A].

The design procedure can be decomposed into two broad subprocedures, the "general design" and the "detailed design". Broadly speaking, the general design defines the simulator and user layers, while the detailed design refines these layers and defines the task layer.

The following steps comprise the design procedure:

1. General Design
 - A. Task Description
 - B. Environment Description
 - C. Knowledge Domain Description
 - D. User Behavior Description
2. Detailed Design
 - A. Simulator Design
 - B. Context-level Micro-expert Design
 - C. Micro-expert Design

The discussion in this chapter is somewhat brief, as it is meant to serve as an overview of the entire design procedure. Many of the design concepts are more easily followed in the context of an example, and are presented more fully in Chapters Five and Six. A more thorough discussion of many of these steps may be found in Appendix D, Planning Task Simulator Design Guide.

4.2. GENERAL DESIGN

4.2.1. Task Description

The task description defines the task to be performed by the simulator, identifies the real-world individual or group responsible for performing the task (the task specialist), and defines the inputs and outputs to the task specialist.

The input definition details the resources and information sources available to the task specialist for the performance of his task; the output definition details the content and destination of all required reports.

4.2.2. Environment Description

Describing the planning environment consists of identifying the entities superior, subordinate, and collateral to the responsible individual performing the given task. Note that this step identifies the individual's chain of command, as well as any coordinating units that affect the performance of the task. Additionally, the environment description includes the description of the communications model to be used in the simulator.

4.2.3. Knowledge Domain Description

The primary knowledge used in planning is in the form of decision heuristics. These are procedural and are naturally represented by behaviors in ROSS. The declarative knowledge is the data used to drive the decisions (hence, data-driven). The data needed to

perform the task is determined by the situation, i.e. the proper situation assessment is the collection of the data required to make an informed decision.

Describing the required knowledge domains consists of identifying what data is needed to perform the given task. The description should include whether the data is expected to change with time and whether the data is available to more than one individual. The data should also be classified by type: data can be facts, constraints, representations of the environment, objectives, descriptions of the goals, etc. The way the data is used by the procedural knowledge will determine the method of storing the data in the extended ROSS environment.

4.2.4. User Behavior Description

Describing the user's behaviors consists of identifying what functions the user will have to perform during the simulation (if any). Typical functions would include furnishing certain required input information and possibly critiquing the simulator output. For particularly difficult portions of the task, the user may also be called upon to perform certain data processing or reduction tasks. One example of this type of behavior

exists in SAMPLE, where the user furnishes terrain data to the simulator based upon reading a standard map.

4.3. DETAILED DESIGN

4.3.1. Simulator Design

During the simulator design phase, the simulator actors are identified, actor relationships are defined, and the task-specific "controller actor" and "designer actor" are specified. The controller actor is responsible for the proper sequencing of the simulator, while the designer actor is responsible for the dynamic creation of the required instances of the simulator actors. Also, any extensions to the task simulator system functions are incorporated during this phase.

4.3.2. Context-Level Micro-Expert Design

In the context-level design of the micro-expert, the micro-expert task algorithm is defined. The algorithm should be explicit enough to allow a non-expert to follow the steps and produce results comparable to those of the expert; therefore, the algorithm must embody the expert

knowledge required to perform the task (which is normally provided through specialty training and experience). Much common-sense (non-domain-specific) knowledge will be embodied in the heuristics of the task algorithm. However, care must be taken to also include the necessary "world-knowledge" that is taken for granted by the human planner.

4.3.3. Micro-Expert Design

This step consists of a top-down refinement of the task algorithm into ROSS code. Given that the algorithm is sufficiently explicit, the environment is properly defined, and the required data are properly represented, this step should be relatively straightforward. The use and classifications of the data will drive the selection of the representation mechanisms, and the inherent modularity and transparency of ROSS makes coding the algorithm quite facile. Additionally, the task does not have to be completely defined prior to implementation of portions of the algorithm. ROSS supports the dynamic creation of actors and dynamic addition of behaviors; so, through the use of "stubs", the development and refinement of the micro-expert can be done in an incremental, evolutionary fashion.

CHAPTER FIVE

SAMPLE GENERAL DESIGN

5.1. INTRODUCTION

The sample system we have implemented is called SAMPLE (Surface-to-Air Missile Placement Expert). The purpose of SAMPLE is to demonstrate how the design procedure in Chapter Four may be applied to the simulation of planning tasks. The micro-expert developed for this thesis is not complete, in the sense that it accomplishes all of the functions that a human expert does in the solution of the given task. In fact, it operates on only a limited number of situations that the human expert would have to contend with. However, we believe that the development is complete enough to illustrate the issues involved in planning task simulation. We feel that the extension of the micro-expert into a complete simulator is a problem in design refinement rather than definition or specification.

In another sense, the micro-expert has been over-designed, in that some design features have been included that are not explicitly needed in the solution of the given task. For example, the communications model incorporated into the simulator environment is not

necessary for the solution of the task. The reason that such features have been included is to demonstrate how the features can be incorporated into a complete system.

Solution of the example task could have been accomplished through the use of an optimization routine, along the lines of an operations research project. However, we chose to develop SAMPLE to act in a manner similar to the human planner to take advantage of doctrine and the heuristics they embody. Either approach may be used for the construction of micro-experts. The decision should be based on the application of the micro-expert, and the use of its output by other experts of the simulator.

Our primary goals in developing SAMPLE were to make it simple enough to clearly illustrate our design philosophy, and yet complex enough to demonstrate that real-world problems can be solved by such a system.

5.2. TASK DESCRIPTION

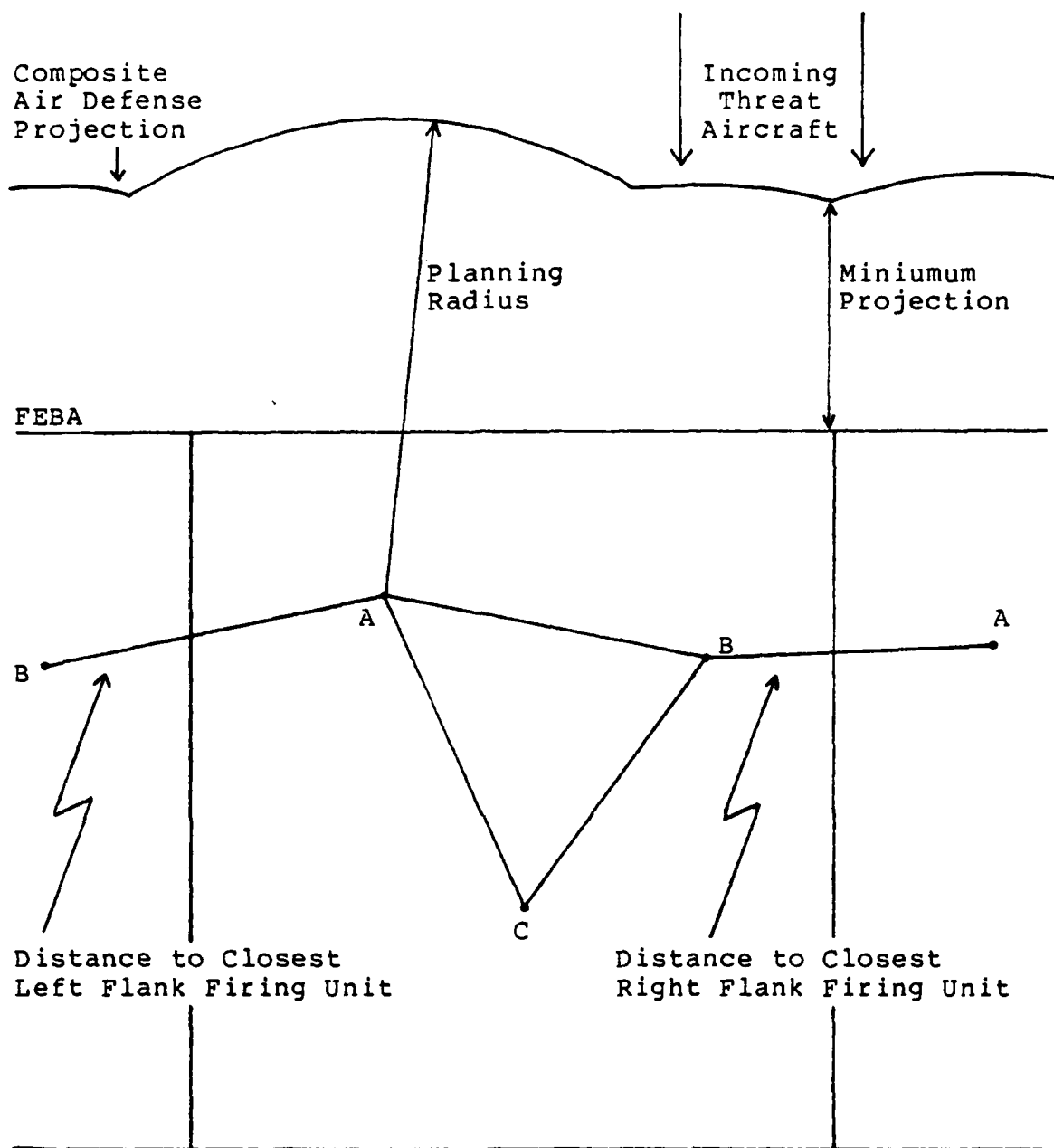
The task that SAMPLE will perform is that of choosing the deployment locations of the firing and headquarters batteries of an Improved-Hawk (I-Hawk) triad battalion. (We will use the terms Hawk and I-Hawk interchangeably,

since only the missile system characteristics differ.) We have limited the task domain further by specifying that the site selection will only be done for the case where the battalion is in direct support of a division in a defensive posture, the supported division is flanked on both sides, and the division area is rectangular in shape.

The real-world individual responsible for this task is the I-Hawk battalion commander. Figure V-1 pictorially describes the site selection task. The commander has three firing batteries, each consisting of one base platoon and two deployable platoons, at his disposal. He places these firing units on suitable terrain in such a way as to provide an air defense "umbrella" for the supported division against low-to-medium altitude threat aircraft.

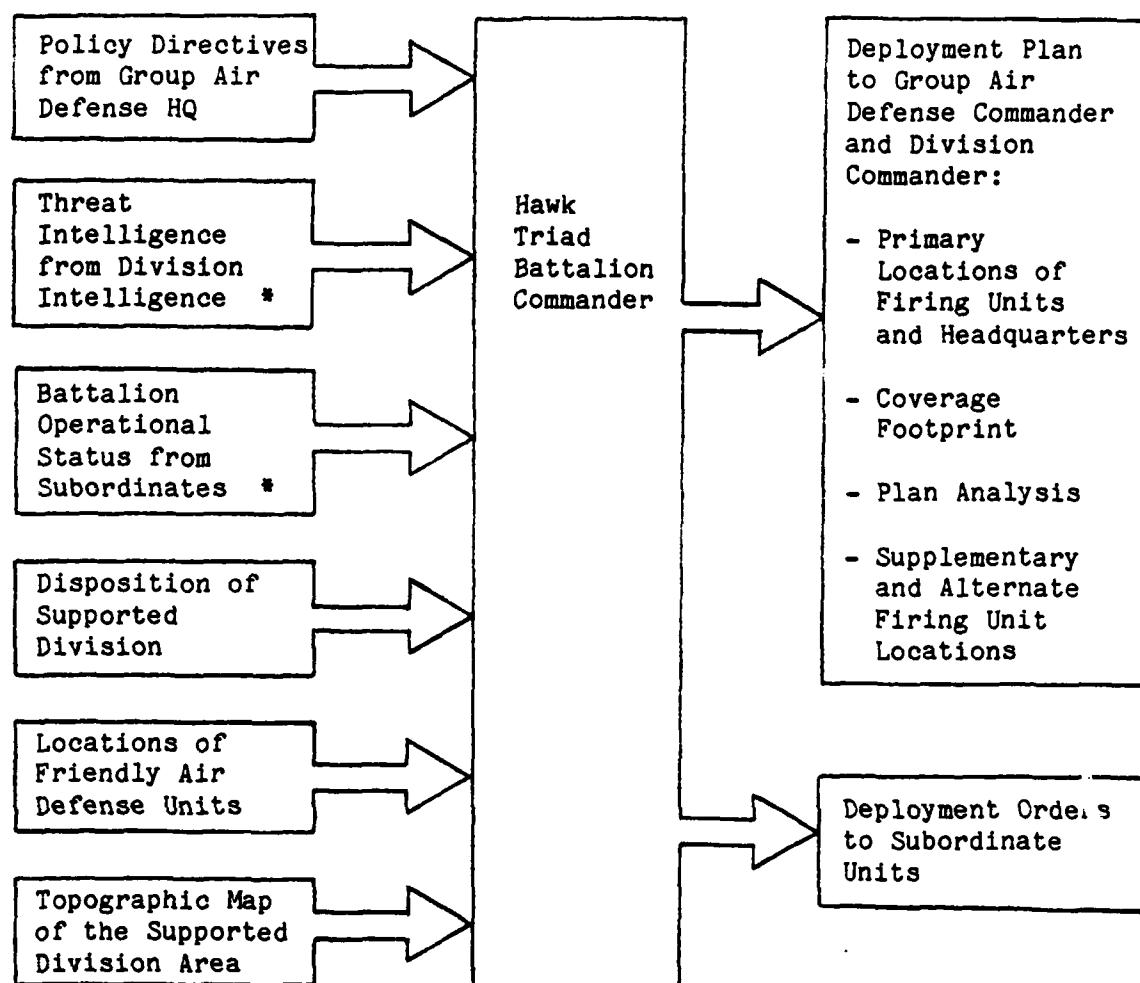
Figure V-2 summarizes the task inputs and outputs. Army field manual FM 44-90 describes the required input information for the performance of this task, consisting of:

1. General guidelines and policy directives from the Group Air Defense Headquarters.
2. Threat intelligence information from division intelligence.
3. Battalion operational status from subordinate firing batteries.
4. Division posture and objectives information from the supported division.
5. Locations of friendly air defense units.
6. A topographical map of the supported division area.



Capital Letters Denote Firing Batteries

Figure V-1. Typical Deployment Scenario



* - Not Used By SAMPLE

Figure V-2. Task Inputs and Outputs for the Site Selection Task

Not all of this information is used by the SAMPLE task algorithm. Items 2, 3, and 4 are included as "typical" values, and not as variable input values. Item 1 consists of system characteristics and mission objectives. The system characteristics determine the parameters of a usable site and the planning radius (the effective range of the radar and missile system). The mission objectives include the commanded projection (extent of coverage into enemy territory) and adjacency (distance to neighboring SAM sites to provide overlapping coverage). Appendix B, "Site Selection Objectives", describes how the site selection procedure changes in response to changes in this input information.

The required outputs consist of a deployment plan and deployment orders to the subordinate firing batteries. The deployment plan is forwarded to the Group Air Defense Commander and the supported division commander. Information in the deployment plan includes the locations of the three firing batteries, the locations of any deployed platoons, a radar coverage footprint, and an analysis of the planned deployment. The analysis includes reasons why any mission constraints were not met by the deployment plan, and the reasons for deploying deployable platoons.

SAMPLE does not include the coverage footprint in its output to the user, nor the locations of deployable platoons.

5.3. ENVIRONMENT DESCRIPTION

Figure V-3 details the organizational environment in which the Hawk battalion commander performs his task. The superior unit of the Hawk battalion is the Group Air Defense Headquarters, and the subordinate units are the three firing batteries assigned to the triad battalion. Because the Hawk battalion is in direct support of a division, the Hawk commander is also responsible to the division commander; this is modeled in SAMPLE by providing a command channel between the division commander and the Hawk battalion commander.

The Hawk battalion commander is also responsible for maintaining coordination with the supported division's organic air defense chief, and has collateral (information only) access to the "sibling" Hawk battalions in support of the divisions on either side of the supported division.

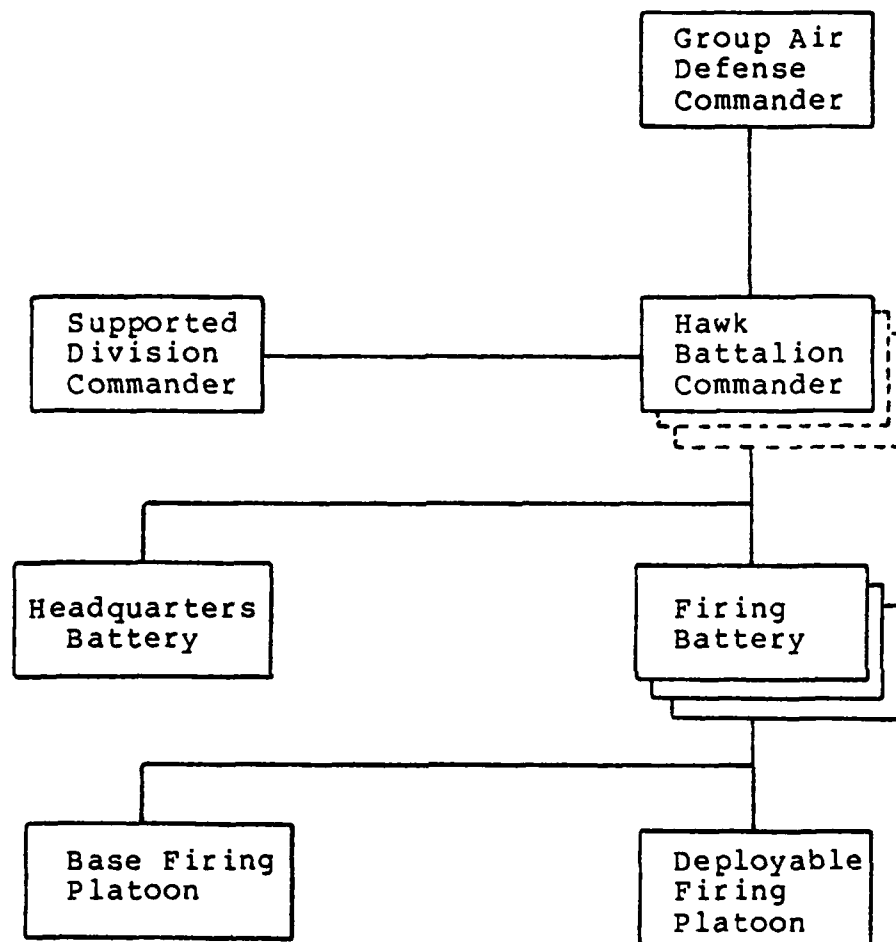
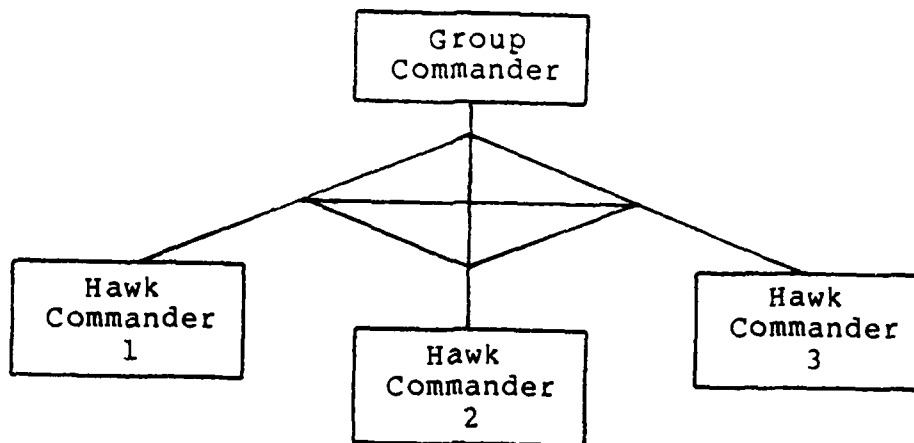


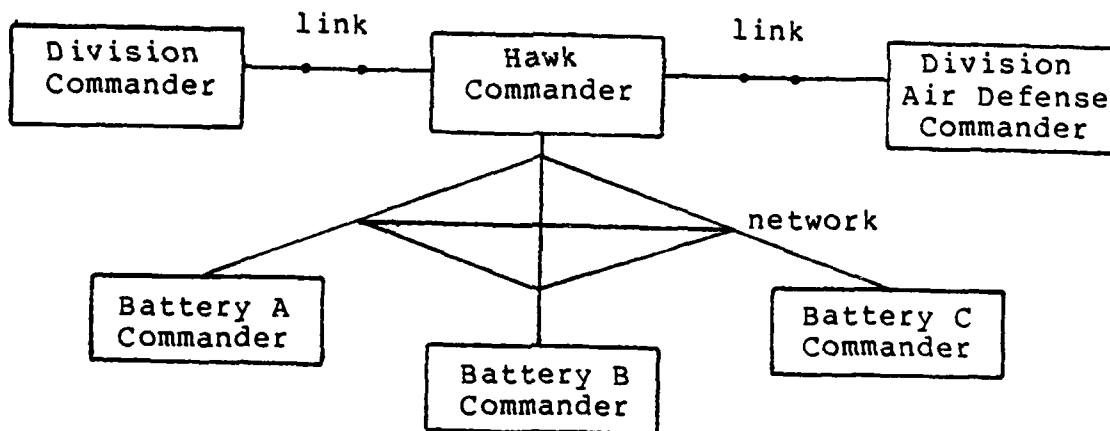
Figure V-3. Organizational Structure of a Triad Battalion in Direct Support of a Division

The communications model used in SAMPLE is illustrated in Figure V-4. Each Hawk battalion commander has four communications channels assigned to him, two networks and two links. One network connects the members of the Air Defense Group (the Group commander and his subordinate Hawk commanders), while the other connects each Hawk battalion member (the Hawk commander and his subordinate battery commanders). The two links establish contact with the supported division: one is a command link to the supported division commander, while the other is for coordination purposes to the division air defense chief.

We feel that this type of communications model could easily be extended to realistically simulate the command, control, and communications behavior exhibited by an actual Hawk battalion. For example, by assigning a frequency attribute to each comm node, and by implementing a communications protocol as part of the ROSS behaviors of the nodes, the messages passed among the leading actors would closely resemble the message traffic of an actual battalion. Also, dynamic removal of a comm node from the SAMPLE environment would correspond to the destruction of an actual communications capability, and duplicate nodes could be added to the system to model real-world redundancies. Behaviors to model the effects of saturation and jamming could also be incorporated.



[A] Group Network



[B] Other Communications Channels

Figure V-4. SAMPLE Communications Model

5.4. KNOWLEDGE DOMAIN DESCRIPTION

5.4.1. Areas of Expertise

The Hawk commander needs expertise in the following areas to effectively place his firing units:

1. Hawk system characteristics and parameters.
2. Situation assessment.
3. Terrain.
4. Deployment objectives and doctrine.

System characteristics refer to the operating limits of the Hawk surface-to-air missile system, such as maximum range against low-to-medium altitude targets, maximum allowable site gradient, minimum required area to place a firing unit, etc.

Situation assessment refers to the ability of the commander to react to the requirements of his current situation and come up with a deployment plan that meets mission requirements. Specific abilities of the commander include modifying the expected system range based on the expected threat, incorporating the constraints of superiors into his plan, and choosing the appropriate strategy based on the available terrain and assigned mission.

The commander must know how terrain limits and enhances the capabilities of his weapon system. Also, he must know how to read a military map and decide whether a given location is suitable for placing a firing unit.

Finally, the commander must know his mission, his short-term objectives, deployment doctrine, and how these factors drive his selection of firing sites. Established doctrine is what provides the commander with the decision heuristics used in accomplishing his task, while the mission and the situation objectives guide the commander in choosing the proper heuristic to apply in a given situation.

Not all of these areas of expertise are fully implemented in SAMPLE. For example, our task algorithm is based on a constant system range (known as the "planning radius") that does not take into account the expected threat's altitude or electronic countermeasures. However, we have used a conservative, or worst-case, value of this parameter, so that the algorithm solves the more highly-constrained case. Additionally, we have kept a human in the loop to provide abstracted terrain data to the system, as opposed to automating the map-reading function.

5.4.2. Knowledge Domains

The Hawk commander's expertise can be classified into two major knowledge domains: the knowledge needed to identify acceptable sites, and the knowledge needed to evaluate and choose the "best" sites. An acceptable site must be near an accessible road. The terrain must be fairly flat over an area of approximately 400 by 400 meters. The ground cover must be light enough to allow the equipment to be brought in with wheeled vehicles and maneuvered. Most importantly, a site must be in a location where it meets the mission objectives of projection and adjacency and yet retain a degree of security from enemy artillery.

The evaluation and selection procedure requires knowledge of the interactions of the candidate sites. Sites are evaluated on the degree of acceptability (based on the criteria listed above), the dominance of the site (height, in order to project over terrain features that would mask the radar), and the degree of overlap between sites. One desirable condition is to have the site within the coverage umbrellas of neighboring sites in order to protect each other from attacking enemy aircraft. The selected sites, therefore, cannot be chosen independently of each other, but must be evaluated as a group.

These two knowledge domains clearly are not used independently of each other. The evaluation process uses knowledge of the acceptability of a site, and sites are not tentatively selected based on the acceptability criteria alone. In particular, acceptable sites may lie in valleys, but would not be examined until all dominant sites were eliminated from consideration. This prevents the commander from selecting tentative locations that would not compete well in the evaluation process. The dominance of a site is an evaluation criteria, not a factor contributing to the acceptability of the site. Both knowledge domains are used interactively by the commander as needed. The process of utilizing the knowledge is, therefore, opportunistic.

5.4.3. Data Representation

System characteristics are static, in that they do not change from one scenario to the next. As such, these characteristics are stored in a read-only database file structured as a collection of Lisp `setq` statements. The situation assessment data are quasi-static, in that the data change from one scenario to the next, but remain constant throughout a scenario. These data are stored in the personal memories of the various leading actors. Terrain data remains constant throughout a scenario (since

we assume that the supported division is not moving), but the focus of attention of the micro-expert changes often. Therefore, the current terrain data are dynamic, and are stored on scratchpads.

Generally speaking, data are grouped and stored as table or list structures. For example, an individual hill is represented as a list: the first element is the location of the hill, the second and third elements are the elevation of the hill and location of the nearest road, respectively.

5.4.4. Heuristics

Two types of heuristics are used in SAMPLE: decision heuristics and case-selection heuristics. Decision heuristics are the rules used to choose locations from a list of possible candidates, and are implemented as Hawk-commander behaviors. Case-selection heuristics are the rules used to determine which decision heuristics to apply in a given situation. Case-selection heuristics may be either implemented as behaviors, or "designed-in" -- implicit in the limited scope of the task algorithm. Limiting the scope of the algorithm simplifies the micro-expert design and implementation, resulting in a less robust system. However, after development of the

limited micro-expert, the case-selection heuristics may be changed to behaviors. The micro-expert should be modular enough to transport directly into the extended system as a specialist in the limited case.

5.5. USER BEHAVIOR SPECIFICATION

The user plays the role of the Hawk battalion commander's superior units, and therefore supplies the scenario definition to the system. Components of the definition include:

1. Degree of overlap required with adjacent Hawk units, and the minimum required radar coverage past the FEBA.
2. Division width.
3. Locations of friendly air defense units.

The user is also responsible for inputting terrain data to the simulator, under the direction of the micro-expert. Map reading behaviors include:

1. Determining how many usable and unusable hills are in a specified region.
2. Determining hill parameters, to include height, location, and the location of the nearest road.
3. Determining the minimum and maximum elevations in a given region.

Complete specifications of the user's interaction with the system is presented in Appendix C, "SAMPLE User's Guide".

CHAPTER SIX

SAMPLE DETAILED DESIGN

6.1. SIMULATOR DESIGN REFINEMENT

Figure VI-1 presents the block diagram of the functional modules used in SAMPLE. Figures VI-2 and VI-3 present these modules as ROSS actors, and illustrate the hierarchical (inheriting) relationships among them. Descendant actors will inherit the behaviors and attributes of their parent(s); as shown in the figures, it is possible for an actor to have more than one parent.

6.1.1. Hierarchical Structure

Given the design procedure established in Chapter Four, all single-task planning simulators will exhibit a high-level hierarchical structure similar to that of Figure VI-2. The "something" actor is resident in the initial ROSS environment and contains the behaviors responsible for the creation of actors, manipulation of attributes, definition of behaviors, etc. The "thing" and "system" actors, as well as the generic descendants of "system," are resident in the extended ROSS environment.

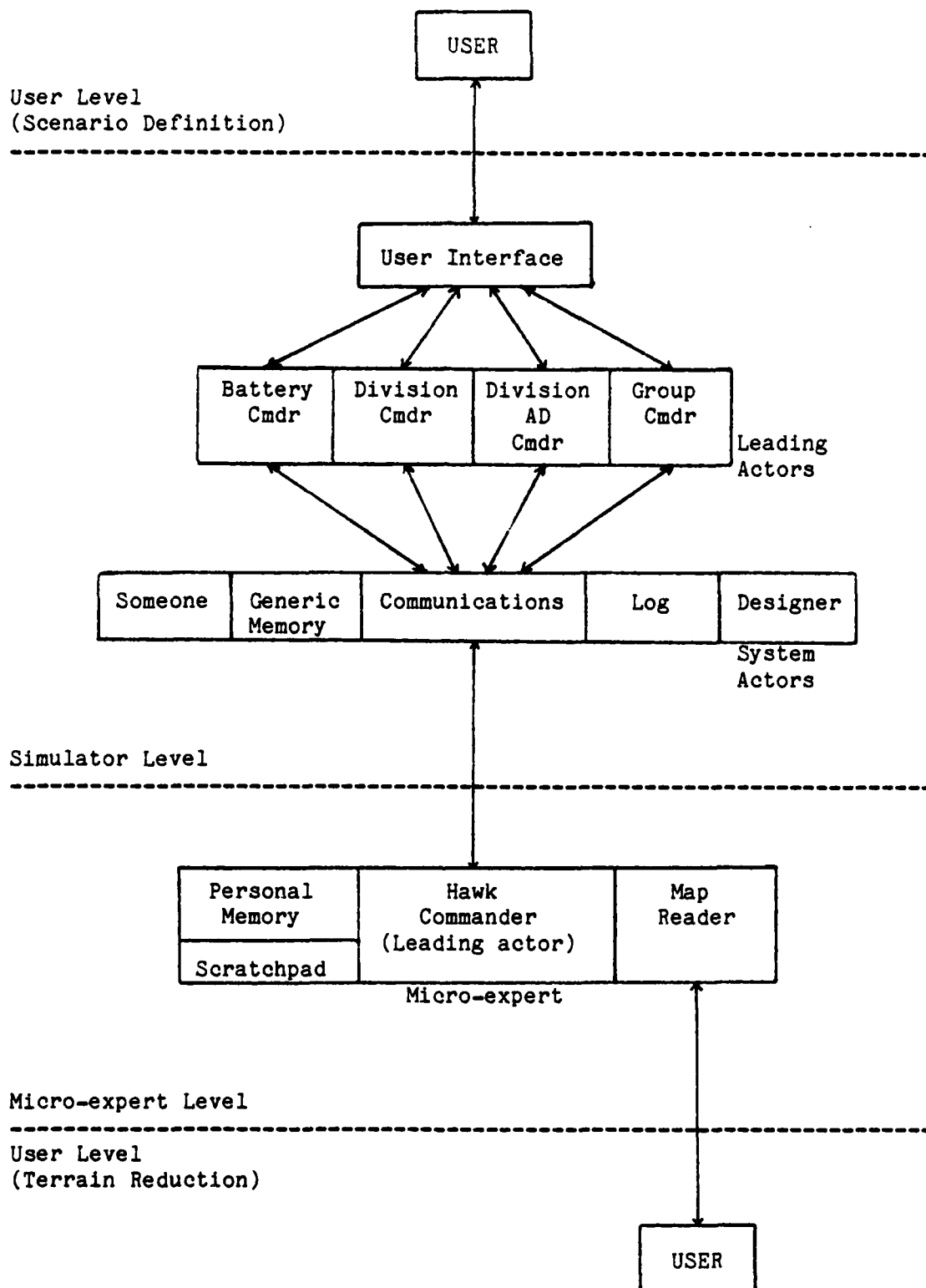


Fig VI-1. SAMPLE Functional Modules.

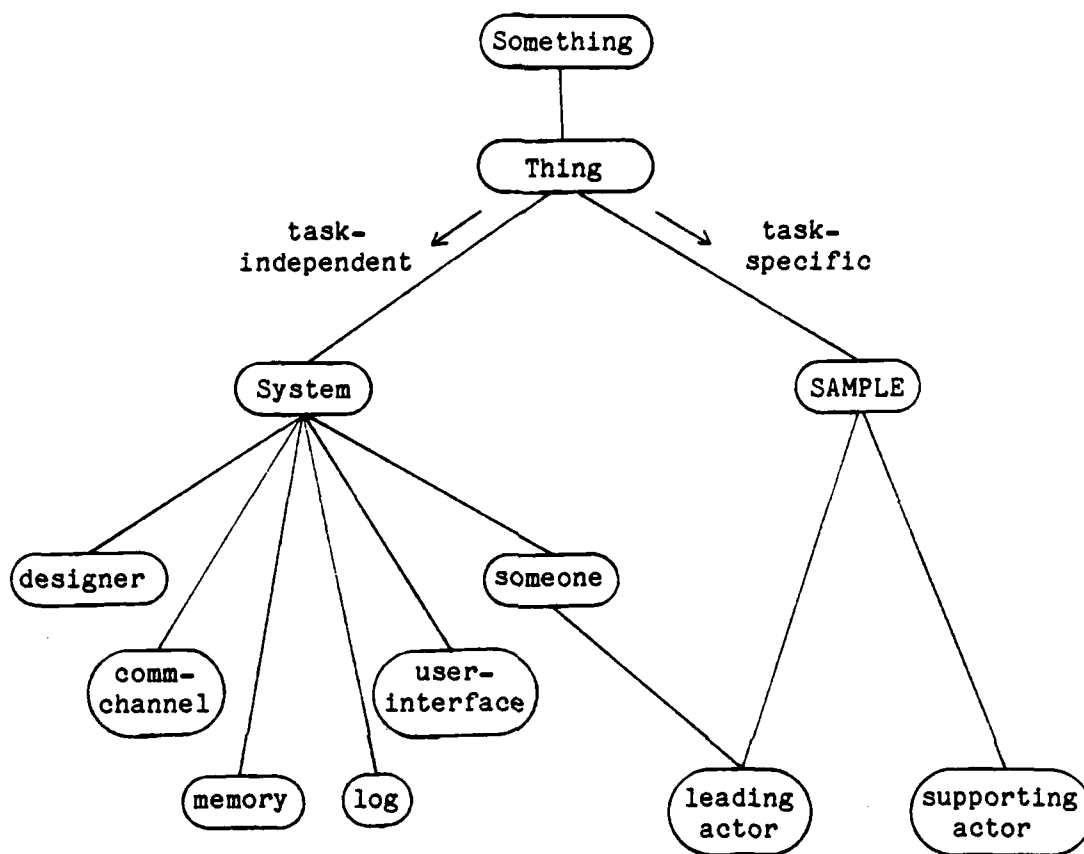
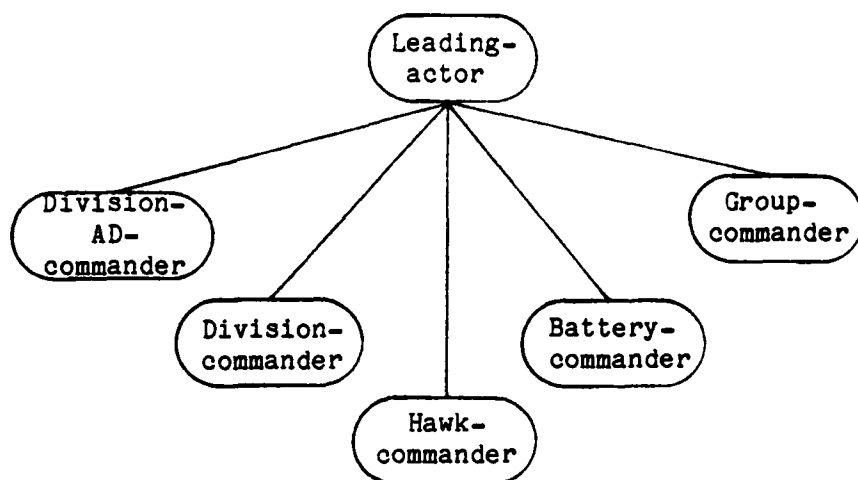
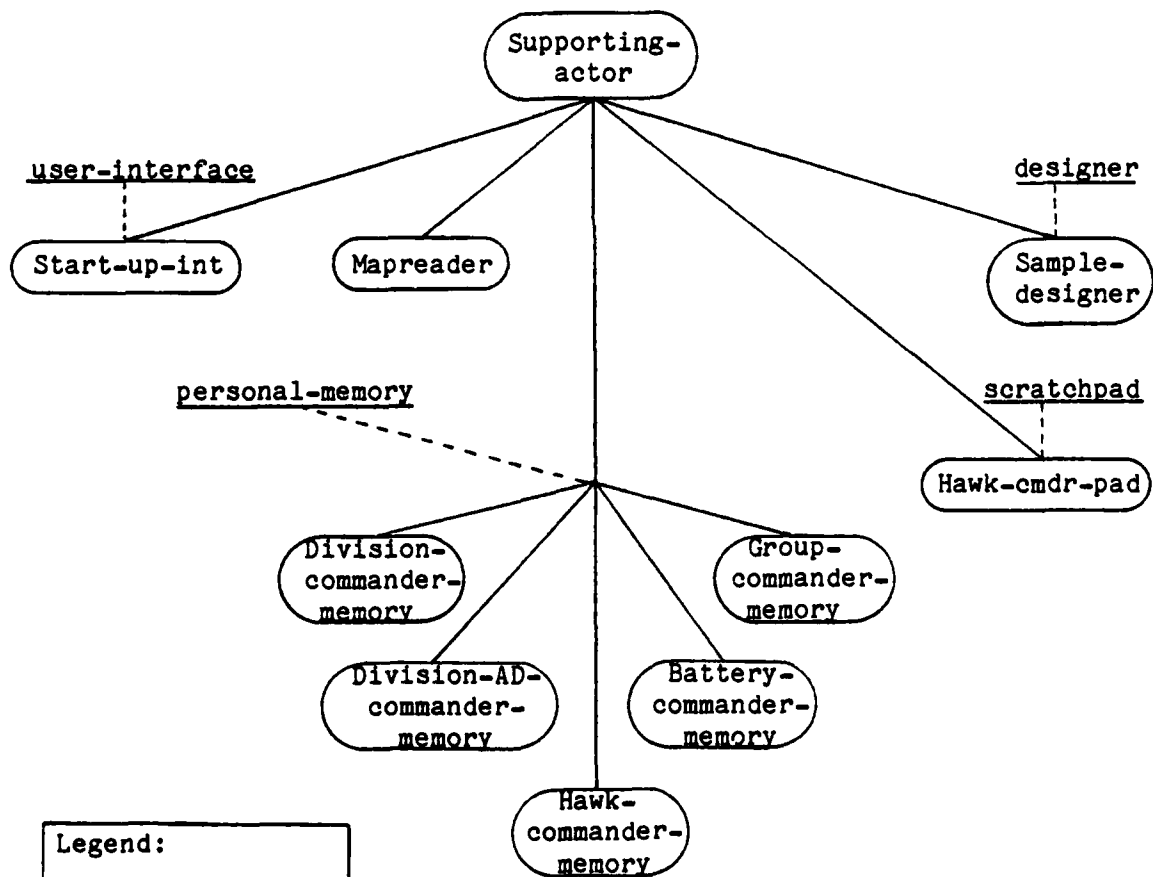


Figure VI-2. SAMPLE Top-level Hierarchy.



Part [A]



Legend:
 --- additional parent

Part [B]

Figure VI-3. Leading and Supporting Actor Hierarchies.

The descendants of the "sample" actor implement the task-specific behaviors performed by the simulator, and are divided into leading and supporting actors (as defined in Chapter Three). The micro-expert simulates the real-world task specialist and is therefore a leading actor (in the sense defined in Chapter Three). The SAMPLE leading actor structure is shown in Figure VI-3, Part [A].

The only SAMPLE leading actor with its own (i.e., non-inherited) behaviors is the "hawk-commander" actor. The required behaviors of the other leading actors are either performed by the user as part of the scenario definition, or are handled by the inherited behaviors of the "someone" actor.

6.1.2. Supporting-Actor Functions

Figure VI-3, Part [B] illustrates the supporting actor structure.

"Start-up-int" is responsible for conducting the scenario definition dialogue with the user.

"Mapreader" is logically a component part of the micro-expert and will be discussed in detail in later sections. Briefly stated, its function is to translate

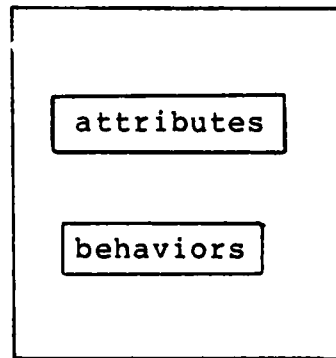
symbolic requests for terrain data into specific user instructions and to relay the answers back to the micro-expert.

"Sample-designer" is responsible for creating dynamic actor instances. It translates symbolic requests such as "Create an air defense group of 5 members" into the appropriate system messages needed for the creation of the group members.

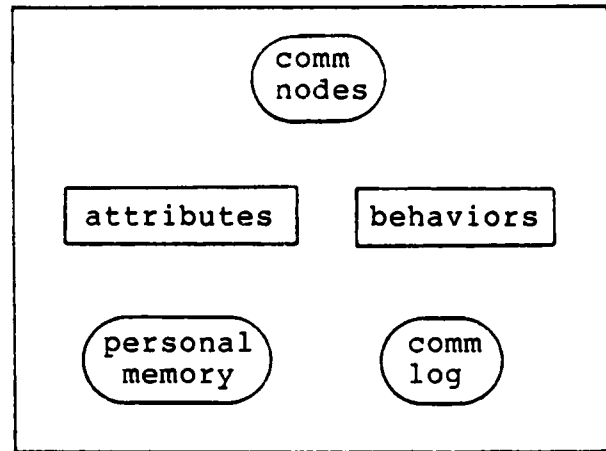
The memory actors (hawk-cmdr-pad and the leading actors' personal memories) are responsible for data storage and retrieval. Their behaviors are inherited from system actors, but they contain task-specific attributes -- symbolic data definitions and initialization information.

6.1.3. Actor Structure

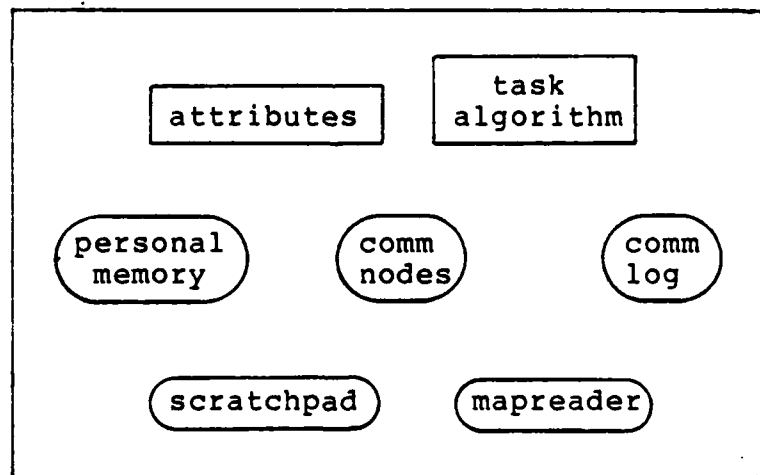
Figures VI-2 and VI-3 detailed the hierarchical structure of the SAMPLE simulator actors from an inheritance point of view. In an object-oriented language, complex functions are most easily implemented through the interaction of distinct but logically related actors, each performing a portion of the function. Figure VI-4 shows the relationships among the SAMPLE actors.



Part [A] System or Supporting Actor



Part [B] Leading Actor



Part [C] Micro-expert

Figure VI-4. Logical Actor Structure.

The structure of a system or supporting actor is shown in Part [A] of the figure. These actors have the simplest logical structure, being composed of the attributes and behaviors included as part of the actor definition.

Leading actors are slightly more complicated, and their structure is shown in Part [B]. In addition to the attributes and behaviors of the leading actor definition, a personal memory and comm log (which are system actors in their own right) are "assigned" to each leading actor. The purpose of these assigned actors is to extend the data representation capability of the leading actor.

The most complex logical structure is that of the micro-expert, shown in Part [C] of Figure VI-4. Because the micro-expert is a leading actor, it is assigned a personal memory and comm log. In addition, the micro-expert is assigned a scratchpad and mapreader to allow the micro-expert to deal with terrain data. The scratchpad localizes the terrain data representation used by the component actors, while the mapreader extends the procedural knowledge of the micro-expert. The mapreader is responsible for querying either a human being for the needed terrain data (in the case of SAMPLE), or an automated terrain database (in the case of a totally automated simulator). By separating out the mapreader

behaviors into a distinct actor, the behaviors of the hawk-commander actor remain the same in both cases.

6.1.4. Actor Interactions

The focal point of the user interaction is the "Continue" file. This file transmits definition and constraint data requests to the user-interface actor in a symbolic form. The user-interface formats the user query, accepts and validates the input data, and passes the data back to Continue. Based on the scenario definition, the Continue file issues actor creation requests to the sample-designer actor, which creates the actors. Continue then transmits the required scenario definition and constraint data to the leading actors in the form of memory storage messages. Having thus defined the scenario to the leading actors, Continue issues a "start simulation" message to the selected hawk-commander actor.

The message passing among simulator leading actors is regulated by the communications actors assigned to them. Having received "start" messages from Continue, the group and division commanders transmit their respective information messages to the micro-expert through their assigned communications channels. During the course of the simulation, the micro-expert will query his adjacent

hawk commanders and the division air defense commander for their locations. These requests and their responses are again routed through the assigned channels.

6.1.5. System Files

Several task-specific system files serve to augment the simulator actors:

1. "Sample-fns" is a compilation of the Franz Lisp functions called by the simulator actor behaviors. While these functions were developed to perform task-specific operations, we hope that many can be used in other task domains as well.
2. "Sample-abbrevs" contains the ROSS abbreviation definitions used by SAMPLE. The abbreviation package is not resident in ROSS, but was included in the system installed on the AFIT SSC. This package substantially improves the readability and transparency of the code. Again, many of the abbreviations are appropriate to any ROSS implementation; in fact, most of the ones included in SAMPLE were originally defined for SWIRL, an air battle simulator written in ROSS and developed at RAND (McArthur, 1982).
3. "Sample-design" contains the definitions of the global variables used in SAMPLE. It is read in by the Lisp reader; therefore, the format of this file is tightly constrained. The purpose of this file is to localize static system variables and thereby permit easy modification to the simulator.
4. "Continue" is used to "boot" the simulator from within the initial ROSS environment. It contains the instructions needed for the dynamic creation of actor instances, and the messages needed to initialize the created actors.

6.2. CONTEXT-LEVEL MICRO-EXPERT DESIGN

The site selection task performed by a Hawk commander in direct support of a division is described in general in Army Field Manual FM 44-90. The following pseudocode presents the first-level decomposition of the task algorithm used in SAMPLE:

1. Conduct preliminary evaluation.
2. Determine forward regions of interest.
3. Choose locations of forward batteries.
If solution is still feasible then
4. Choose location of rear battery.
5. Analyze the plan.
6. Choose locations of deployable platoons.
7. Analyze the plan.
8. Choose location of headquarters battery.
9. Issue deployment warning.
10. Report.

Only steps 1, 2, 3, and 10 are implemented in the current version of the SAMPLE simulator. For the most part, the behaviors required in the remaining steps are modifications to the behaviors used in the implemented steps. Note that if the given situation is not feasible (that is, the algorithm fails to find suitable forward locations), steps 4 through 9 are not attempted -- rather, the user is notified immediately.

We would like to emphasize that the task algorithm presented here is our interpretation of the information presented in the field manual; the algorithm has not been examined by an air defense expert for either completeness nor accuracy. Because of the lack of expert review, ease

of modification and clarity of presentation have been our primary design goals. We feel that the task algorithm we have developed clearly demonstrates the potential of the proposed design methodology, and we believe that the extension of this algorithm to a complete and accurate representation of human task performance would be relatively straightforward.

6.3. MICRO-EXPERT DESIGN

6.3.1. Micro-Expert Component Interaction

Figure VI-5 defines the interactions among the micro-expert components that occur during the execution of the task algorithm. The Hawk-commander actor generates symbolic data requests and passes these requests to the mapreader actor. The mapreader then prompts the user for the required data and either passes the data directly to the Hawk-commander or updates the scratchpad. The scratchpad localizes the terrain data, and provides data storage and retrieval functions. Upon creation, the scratchpad tables are initialized with the data contained in the "sample-design" file described in Section 6.1.5. The Hawk-commander actor uses the information contained in the scratchpad to drive the decision heuristics implemented in the task algorithm behaviors.

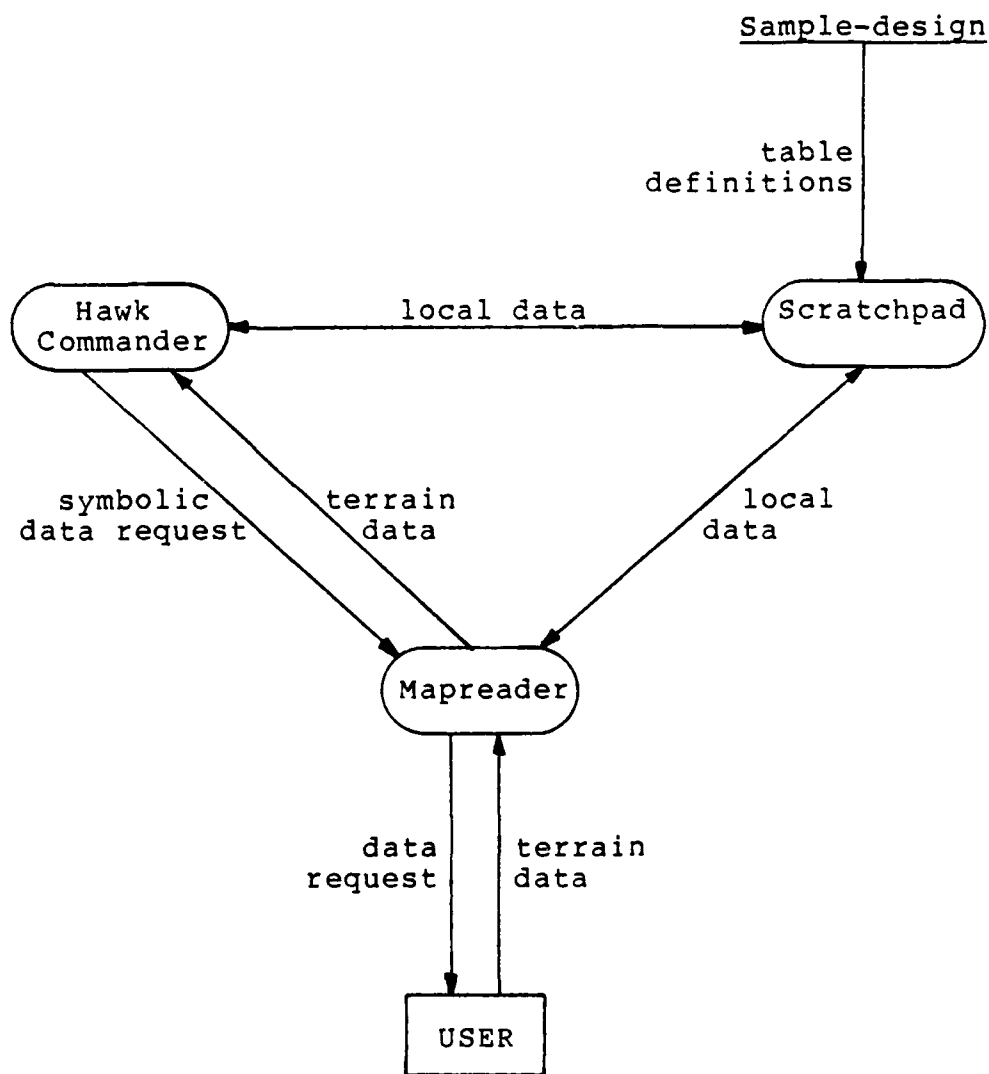


Figure VI-5. Micro-Expert Component Interaction.

6.3.2. Level Two Task Decomposition

Step One, "conduct preliminary evaluation", is further decomposed into the following steps:

- 1.1. Collect required input information.
- 1.2. Reallocate battalion assets if necessary.
- 1.3. Determine planning radius.

During step 1.1, the Hawk-commander actor issues location queries to its adjacent battalions on either side. The responses are then stored on the assigned scratchpad. Also during this step, copies of the data from personal-memory (entered earlier) are stored on the scratchpad.

Step 1.2 is not implemented; if it were, the Hawk-commander would adjust battalion assets so as to equalize battalion assets among the three firing batteries. For example, if one firing battery had only one of three missile launchers operable, then a launcher from a full-strength battery would be assigned to the weaker battery.

During step 1.3, values of the planning radius, mutual support radius, and overlapping fire radius are computed and stored on the scratchpad. (Appendix B describes the definitions and significance of these values.)

Step Two, "determine forward regions of interest", has two sub-behaviors:

- 2.1. Compute region points.
- 2.2. Determine forward regions.

The real-world Hawk commander knows that, in the absence of other constraints, firing units would be placed symmetrically within the division area. This symmetric placement would yield forward batteries located one-fourth of the division-width from the division edges and out of range of threat medium field artillery (about 12 kilometers from the FEBA). Knowing the desired placement strategy, the commander looks for possible deployment sites near the optimum points. This behavior has been modeled in SAMPLE through the use of regions: based on input constraints, division width, locations of adjacent units, and the planning radius, several regions are defined in terms of the potential suitability of sites located within the region.

Step 2.1 computes the values of the points used to define these regions, while step 2.2 constructs the representation of the regions in terms of the points and stores this representation on the scratchpad. By ordering the search for candidate sites according to the potential values of the various regions, the "look for sites near here" behavior of the real-world Hawk commander is

simulated. For example, if several good candidates are identified in the region nearest the optimum point, then the algorithm can avoid looking for candidates in inferior regions. The use of regions does for the algorithm what experience does for the real-world commander: avoidance of an exhaustive search of the entire division area.

Step Three is broken down into the following sub-steps:

- 3.1. Get the division characteristics.
 - If the division is flat
 - then 3.2. Choose candidates using the terrain-free rules.
 - else 3.3. Choose candidates using the normal rules.
- 3.4. Determine best and alternate site pairs.

The purpose of step 3.1 is to determine whether the division area is flat. Selection of deployment sites when the terrain is flat is treated as a special case; since all locations are equally dominant, the decision heuristics change and the definition of candidate sites changes. The current version of SAMPLE does not implement step 3.2.

For each side of the forward division area, step 3.3 decomposes into:

- Loop for each region do
 - 3.3.1. Get candidates in the region.
 - 3.3.2. Score the candidates.
 - 3.3.3. Trim the candidates.

Step 3.3.1 is performed by the mapreader -- the region is described to the user, the terrain data is prompted for, and the user's response is stored on the scratchpad. The scoring procedure of step 3.3.2 is described in Appendix B, Site Selection Objectives, as is the procedure for determining best and alternate pairs (step 3.4).

Candidate trimming is a heuristic device that serves to reduce the amount of storage and computation required to execute the algorithm. After each addition to the list of candidates, candidates with higher (worse) scores and lower elevations are eliminated from further consideration. However, the trimming is not performed until the number of candidates exceeds five.

If at any point a candidate with a score of zero is found (that is, the site is optimum with respect to all selection constraints), then only higher sites will be added to the candidate list (since only higher sites could produce better sites).

Step 10, "report", consists of formatting and passing the candidate pair data and alternatives to the user. Results of an example scenario are presented in tabular form in Chapter Seven, Summary.

CHAPTER SEVEN

SUMMARY

7.1. INTRODUCTION

The purpose of this chapter is to present what we feel are the most significant results of this thesis effort. An example scenario definition and terrain data table are presented first, along with the simulator generated site selections for the forward firing batteries. A short discussion of how the simulator constraint representation resulted in the given solution is followed by our projection of the potential beneficiaries of this type of planning task simulator. The chapter closes with our recommendations of further research needed to realize these potential benefits.

7.2. TASK ALGORITHM RESULTS

Table VII-1 defines the scenario used for one example session. An explanation of the terms used and values allowed can be found in Appendix C, SAMPLE User's Guide. Table VII-2 presents the parameters of the usable candidate sites in the forward division area. The unusable hills input during the session are not tabulated

herein. Table VII-3 shows the scoring of the individual sites, and Table VII-4 presents the mutual scoring of the interaction of site pairs. Appendix B, Site Selection Objectives, contains the explanation of the site scoring procedure and the metrics used.

Table VII-1. Scenario Definition.

Parameter	Value
Projection constraint	12 kilometers
Adjacency constraint	100% of planning radius
Division width	48 kilometers
Left-adjacent location	(-12 -12)
Right-adjacent location	(62 -14)

Table VII-2. Candidate Sites.

Hill	Location	Elevation	Location of Nearest Road
Left-1	(12 -12)	450	(12 -12.5)
Left-2	(14 -14)	520	(15 -15)
Left-3	(13 -8.5)	480	(12 -8.5)
Right-1	(38 -12)	620	(38 -12)
Right-2	(42 -15)	480	(40 -16)
Right-3	(47 -18)	430	(47 -18.1)

Table VII-3. Site Scoring.

Constraint	Left Hill			Right Hill		
	1	2	3	1	2	3
Adjacency	0	0	0	0	0	0
Projection	0	0	0	0	0	0
Safety	1	0	2	1	0	0
Accessibility	2	3	3	0	3	0
Optimality	0	1	0	1	2	3
Masking	2	0	0	0	0	0
Total	5	4	5	2	5	3

Table VII-4. Pair Scoring.

Hill Pair	Support Class	Total Pair Score
Left-2 / Right-1	1	8
Left-1 / Right-1	1	9
Left-3 / Right-1	1	9
Left-2 / Right-3	2	11
Left-2 / Right-2	1	11
Left-3 / Right-2	1	12
Left-1 / Right-3	2	12
Left-3 / Right-3	2	12
Left-1 / Right-2	2	14

The terrain abstraction shown as Table VII-2 was not derived from an actual map; the values used were chosen to illustrate concepts. For example, the hills on the left side are roughly equivalent, while those on the right show a greater range of scores.

Note that hill Right-1 is included in the three best scoring pairs. We would feel confident in choosing this hill as part of the final deployment plan. The closeness

of the scores of the left-side hills points out the need for reporting alternate selections. The human expert makes the final determination of which pair is "best".

7.3. CONCLUSIONS AND RECOMMENDATIONS

Based on our experiences in the design and implementation of a limited planning task simulator, we have come to the following conclusions:

1. ROSS offers a powerful environment for the development and implementation of planning task simulators.
2. Task simulators incorporating the techniques of Chapter Three, and developed according to the procedure given in Chapter Four, can provide automated systems that are highly modular and highly transparent to the end user.
3. The potential benefits of similar task simulators are great.
4. Much work remains to be done.

7.3.1. ROSS Environment

The power of ROSS, and its suitability for task simulator implementation, derives from the ease with which behavioral structures can be integrated with declarative data structures. By allowing the task specialist actor

(whose behaviors embody the decision rules) access to symbolic data structures (contained in the various memory actors), the designer is free to develop systems capable of simulating complicated human behaviors.

7.3.2. Modularity and Transparency

The modularity of the simulator, as exemplified by the simulator structure presented in earlier chapters, allows the designer to change portions of the system without redesigning the entire simulator. Also, much of the work done in the development of one simulator may be transferred to other simulators with little or no modification.

The transparency of the system is demonstrated by the code developed: much of the code can be easily read, especially the first-level decomposition of actors' behaviors. This near-English syntax, and the natural representation schemes used, allows the development of complex systems that non-computer experts can understand.

7.3.3. Potential Beneficiaries

We believe that the following classes of users could benefit from the development of the proposed task simulators:

1. Battlefield commanders could use the simulator to predict the most probable courses of action of the enemy or as a conventional decision aid to develop decision alternatives.
2. Training academies and war colleges could use the simulator as a training tool to demonstrate friendly and enemy planning methods, and as an environment where experimentation can be performed as part of the instruction.
3. War game developers at the Pentagon could use the simulator as either the enemy or friendly troop commander in a computer simulation. Simulators may even be pitted against each other in such a scenario.
4. The intelligence community could use the simulator as a test-bed to verify the accuracy of predicted enemy planning models. Additionally, the system might be used to examine the effects of erroneous, falsified, or incomplete information -- false information about the enemy on the model, or the effect that falsified information would have on the enemy's decisions.

7.3.4. Recommendations for Future Research

The requirement of a human map reader to reduce and input the terrain data severely limits the current simulator's speed of operation and usefulness. We therefore recommend that the map reading function be totally automated. Of course, complete automation would

require a digital terrain representation of the type discussed in Appendix E.

The utility of the proposed task simulators, particularly as battlefield or training aids, would be greatly enhanced by the addition of a graphics input/output support package and natural language interface. The clarity of the simulator would be improved by the addition of an interactive query mechanism as well.

The power of the proposed simulators would be further enhanced by the implementation of learning mechanisms and the development of a coordination and general planning micro-expert (of the type found in Appendix A).

We have stated many times that we believe that useful task simulators can be developed using the methods and techniques proposed in this thesis. Only further research (extension of the SAMPLE simulator and the development of other simulators) will show the validity of our claim.

BIBLIOGRAPHY

- Alexander, Arthur J. Modeling Soviet Defense Decision-making. Rand Corporation paper prepared for the Naval Postgraduate School/UCLA Center for International and Strategic Affairs Conference on Soviet National Security Decisionmaking, Monterey, California, August 1980. (AD-A103362).
- Davis, Randall. "Expert Systems: Where Are We? And Where Do We Go From Here?", The AI Magazine, 3 (2): pp. 3-22 (Spring 1982).
- Druzhinn, V. V. and D. S. Kontorov. Concept, Algorithm, Decision (A Soviet View). Moscow, USSR: 1972. Translated under the auspices of the United States Air Force.
- Fiegenbaum, Edward A., et al. The Handbook of Artificial Intelligence. Stanford, California: HeurisTech Press, 1981.
- Foderaro, John K. and Keith L. Sklower. The FRANZ LISP Manual. The Regents of the University of California, April 1982.
- Hayes-Roth, Barbara, et al. Human Planning Processes. Rand Corporation report R-2670-ONR prepared for the Office of Naval Research, Arlington, Virginia, December 1980. (AD-A095107).
- Klahr, Philip, et al. SWIRL: Simulating Warfare in the ROSS Language. Rand Corporation Note N-1885-AF, September 1982.
- McArthur, Dave and Philip Klahr. The ROSS Language Manual. Rand Corporation Note N-1854-AF, September 1982.
- Ogawa, Hitoshi, et al. "An Active Frame for the Knowledge Representation", Proceedings of the Sixth International Joint Conference on Artificial Intelligence, pp. 668-675. Tokyo, Japan, 1979.
- Sacerdoti, Earl. "Problem Solving Tactics", Proceedings of the Sixth International Joint Conference on Artificial Intelligence, pp. 1077-1085. Tokyo, Japan, 1979.

Stefik, Mark. "Planning and Meta-Planning", Readings in Artificial Intelligence, pp. 272-286. Palo Alto, California: Tioga Publishing Company, 1981.

US Army Field Manual FM 44-90, Hawk Air Defense Artillery Emplacement. Washington D.C.: US Government Printing Office.

Wilensky, Robert. Planning and Understanding. Reading, Massachusetts: Addison-Wesley Publishing Company, 1983.

Winston, Patrick H. Artificial Intelligence. Reading, Massachusetts: Addison-Wesley Publishing Company, 1977.

APPENDIX A

PLANNING TACTICS

As stated in Chapter 2, a general planner must have at its disposal a set of tactics for decomposing the problem into subtasks, delegating the subtasks to more specialized entities, and resolving the conflicts that appear when integrating the subtask solutions (or proposals) to form the plan. This appendix will present two bodies of thought: the first from the paper "Problem Solving Tactics" (Sacerdoti, 1979) which examines the strategies and tactics of the planning process; the second from Robert Wilensky's book "Planning and Understanding" (Wilensky, 1983) which illustrates a large collection of "meta-entities" to implement his theories of plan generation and understanding.

It should become apparent that while the two approaches differ, the underlying mechanisms are quite similar. In both approaches, subcomponents of the planner work to recognize particular types of goal interactions. The subcomponent is evoked when the interaction occurs, and contributes its "expertise" to the total plan. The difference is in the perspective of the explanation. Earl Sacerdoti presents the basic strategies of automatic problem solving, and tactics to improve the efficiency of

AD-A138 060

A PROPOSED MILITARY PLANNING TASK SIMULATOR USING ROSS
LANGUAGE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB
OH SCHOOL OF ENGINEERING G H GUNSCH ET AL. DEC 83
AFIT/GE/EE/83D-24

2/2

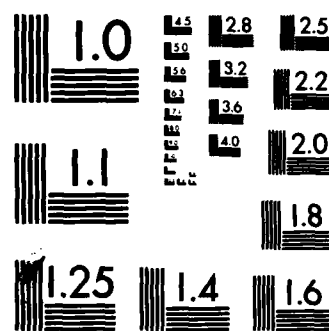
UNCLASSIFIED

F/G 5/1

NL

END

1-1
61-
enc



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

the strategies. Robert Wilensky presents a handful of possible goal interactions, and the micro-entities that handle these interactions. A collection of these micro-entities, with the appropriate control mechanism, can be collectively considered to be a tactic in the Sacerdoti sense.

A.1. STRATEGIES

This section is based on the work by Sacerdoti (Sacerdoti, 1979).

Typically, planning the actions necessary to achieve the goal involves an extensive search through a set of possible solutions. This volume of alternatives, known as the search space, demands that the planner have a number of control strategies to guide the search. The search space is normally described as a tree with the initial state as the root, and branches along every alternative. Hopefully, at least one of the branches leads to the goal state. Another possible tree is one with the goal as the root and the initial state in the upper branches. As one considers all the factors and all the alternatives, the search space becomes enormous and the need for strategies to limit the space becomes clear.

The first strategy Sacerdoti discusses is the "means-ends-analysis" search algorithm used by GPS and STRIPS (Winston, 1977: 130-42) and works as follows. The "difference" between the initial and goal states is determined, and actions are examined to find the one that would most reduce the difference when applied. (The methods of judging the differences and determining suitable metrics with which to make the evaluations are a science unto themselves, and are beyond the scope of this thesis. Many of the factors involved are highly domain-dependent, and so a discussion of them can best be done only in the context of an example.) Once a suitable action is found, it is applied to the initial state, resulting in an intermediate state that is closer to the goal than the initial state is. The process continues in the same manner, moving ever closer to the goal until it is reached.

Sometimes the most promising action cannot be applied directly to the initial state. The preconditions of the action are then chosen to be the subgoal, and a search is made to reach the subgoal state from the initial. This strategy is applied recursively until a sequence of actions can be found that lead from the initial to the original goal state.

The second important search strategy is "backtracking". Backtracking, as described by Sacerdoti is the mirror image of means-ends-analysis. Starting with the goal state, actions are examined to find ones that result in the goal state. The preconditions to these actions are then used as subgoals, and the strategy reapplied to the subgoals until the initial state satisfies the preconditions of an applicable action. At this point a path will have been traced from the initial state through the intermediate states to the goal.

As Sacerdoti points out, neither of these strategies is adequate for complex problems since the search space grows so quickly, especially with multiple goals (conjunctions). Also, ordering of the subgoals may be important, but these strategies have no reason to prefer one conjunction over another. Consequently, the search may try to find a solution to achieve the subgoals in an impossible order. Neither of these strategies distinguishes between the important and the detail. Additional information must be utilized to focus the attention of the problem-solver to that which is critical to the problem. This information is supplied through the use of certain tactics that are applied with the basic strategy to limit the search space.

A.2. TACTICS (Sacerdoti, 1979)

This section is also based on the work by Sacerdoti (Sacerdoti, 1979).

The first and most obvious tactic is called "hierarchical planning". The most important subgoal is tackled first, followed by the next most important, etc. The solution of the important subgoals often leaves the world in a state from which the less important subgoals can easily be reached. This method allows the planner to abstract the situation and the actions, postponing the details and subgoal interactions for later, and allows him to concentrate on the critical decisions without overconstraining the search with influences by less important subgoals. Once the general plan is sketched out, the conflicts and redundancies can be resolved individually, with methods that may have been incompatible with the solution of the main subgoals.

This abstraction can be extended to multiple levels, each level serving as a skeleton to guide the problem-solving process at lower levels. This "hierarchy" of plan skeletons allows complex problems to be broken up into a set of simpler subproblems. However, plans can be created that appear workable at the high level, but which fail in the implementation of the details. The tactic

that resolves this problem, "hierarchical plan repair", checks that all the intended effects of the high-level plan are achieved by the collection of lower-level actions. Only a small number of checks need to be made, due to the heirarchical nature of the plan. The plan can then be repaired, using a variety of methods (Sacerdoti, 1979: 1079). This verification, accomplished as the plan develops, insures that necessary knowledge is not left out by the abstractions. The validity of the plan is then limited by the available world knowledge, rather than the power of the abstraction.

Instead of abstracting the situation as in the heirarchical tactics, "bugging" deliberately makes assumptions that oversimplify the problem of integrating subplans. The planner produces an initial "almost right" plan with bugs in it. However, the bugs are expected and of a limited number of types, and so can be remedied easily. Other names for this tactic are the "debugging approach" and "problem-solving by debugging almost-right plans." The latter was coined by G.J. Sussman, who first employed the bugging technique in his HACKER system (Sussman, 1975).

Principles of these first tactics are incorporated into the tactic "pseudo-reduction" for dealing with goals that include conjunctions (goal 1 AND goal 2). The

conjuncts are treated as subgoals, but they may have the same importance so they cannot be ordered as in the hierarchical planning tactic. Instead, ordering is ignored initially and each conjunct is planned independently. The plans are then integrated in a manner similar to 'bugging'. This tactic reduces the amount of backtracking by avoiding premature commitments to the ordering of the conjuncts and subgoals.

An interesting alternative to actively seeking out a solution to a goal or subgoal is the approach "disproving". Disproving uses a "pessimistic" analysis of the goal to try to show the futility of the endeavor. If any of a set of conjunctive subgoals is found impossible, then it is of little value to work on the others. Furthermore, if a goal cannot be shown to be futile, the knowledge gained can be used by the other "optimistic" tactics.

A.3. META-PLANNING

This section is purposefully brief, since a proper discussion of the issues of meta-planning would constitute a major section of this thesis. The reader is referred to the book Planning and Understanding by Robert Wilensky for a thorough analysis (Wilensky, 1983).

The fundamental assumption of the book (as well as this thesis) is that planning can be accomplished through the coordinated efforts of specialists. Meta-planning, simply stated, is planning about plans: the processes involved in discerning the goals, delegating tasks, formulating subplans, and integrating the subplans into the final solution. Meta-planning is domain-independent, in that the experts involved are applicable to any planning function. Robert Wilensky adds an extra level of expertise to the traditional meta-planner: his system must also understand the situation in terms of plans. This sets up a two-pronged attack on a problem -- plans can be formulated to meet the specified goals by the planning site of his system, and goals can be deduced from given plans and actions of the players in the scenario. The interplay between these two approaches becomes opportunistic in nature, and the system will seek a solution to the assigned problem in whichever fashion works.

The author goes into great depth to explain various goal interactions and how these are understood by the meta-planning system. The control structures of two systems (PAM -- Plan Applier Mechanism, and PANDORA -- Plan ANalysis with Dynamic Organization, Revision, and Application) are explained, along with some details of

their implementations. The problems that arose with these systems are also discussed.

It is our believe that such an implementation of the meta-planning functions can be accomplished in the simulator environment we have developed. The author used a knowledge representaion language (PEARL) which he developed for the representation of the goal interactions and relationships among "objects". However, PEARL discourages procedural attachment. While this was useful in the development of the meta-entities, it becomes a limiting factor to the growth of an extended system. We hold that ROSS provides an equivalent representation mechanism as PEARL, but also provides full procedural capability.

APPENDIX B
SITE SELECTION OBJECTIVES

B.1. INTRODUCTION

The purpose of this appendix is to present the doctrinal principles guiding the site selection task algorithm and the representation mechanisms used by the simulator system to incorporate these principles. In keeping with our implementation philosophy, the numerical values used in the representation are meant only to exemplify how the incorporation may be handled.

B.2. DOCTRINAL OBJECTIVES

Air defense doctrine defines the following objectives that must be met to ensure adequate air defense:

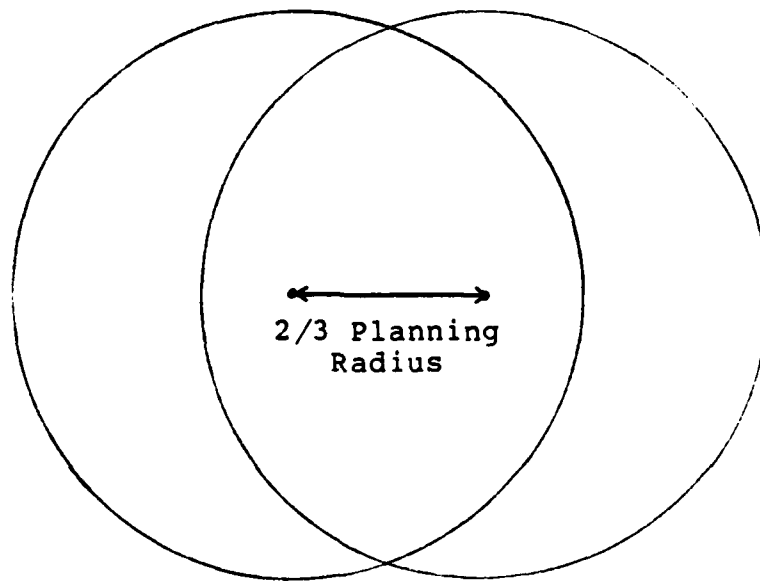
1. Air Defense Projection beyond the FEBA.
2. Balance.
3. Weighted Coverage.
4. Mutual Support.
5. Safety.

The projection objective requires that firing units be placed far enough forward so as to enable engagement of threat aircraft before entering the airspace of the supported division. Balance requires that the entire

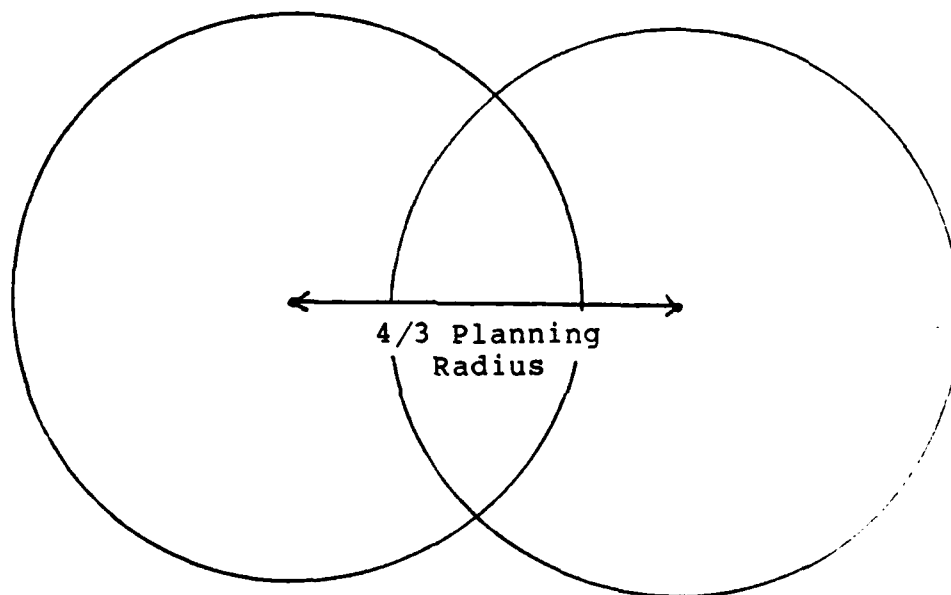
division area be included in the air defense umbrella (also known as "all-around defense"). Weighted coverage requires that likely avenues of attack be given more protection by assigning multiple assets to those avenues. This objective is to be met only to the extent that other objectives are not sacrificed. In the context of the direct support mission, weighted coverage implies that the majority of air defense assets will be placed toward the FEBA. Mutual support requires that firing units be placed so that one unit can cover the blind spots of another unit, if this is possible without violating the other objectives.

Figure B-1, Part [A], illustrates the definition of mutual support. A circle of radius equal to the operating range of the system radar is drawn around the location of the firing unit. If this circle projects past the location of another firing unit by a sufficient margin, then the two units are said to be in mutual support of each other. For the purposes of SAMPLE, mutual support will be established when two firing units are located within two-thirds of the operating range of each other.

Part [B] of Figure B-1 illustrates another important concept in placing firing units--overlapping fire. Two units are said to provide overlapping fires when their respective range circles overlap by a sufficient margin



Part [A] Mutual Support



Part [B] Overlapping Fire

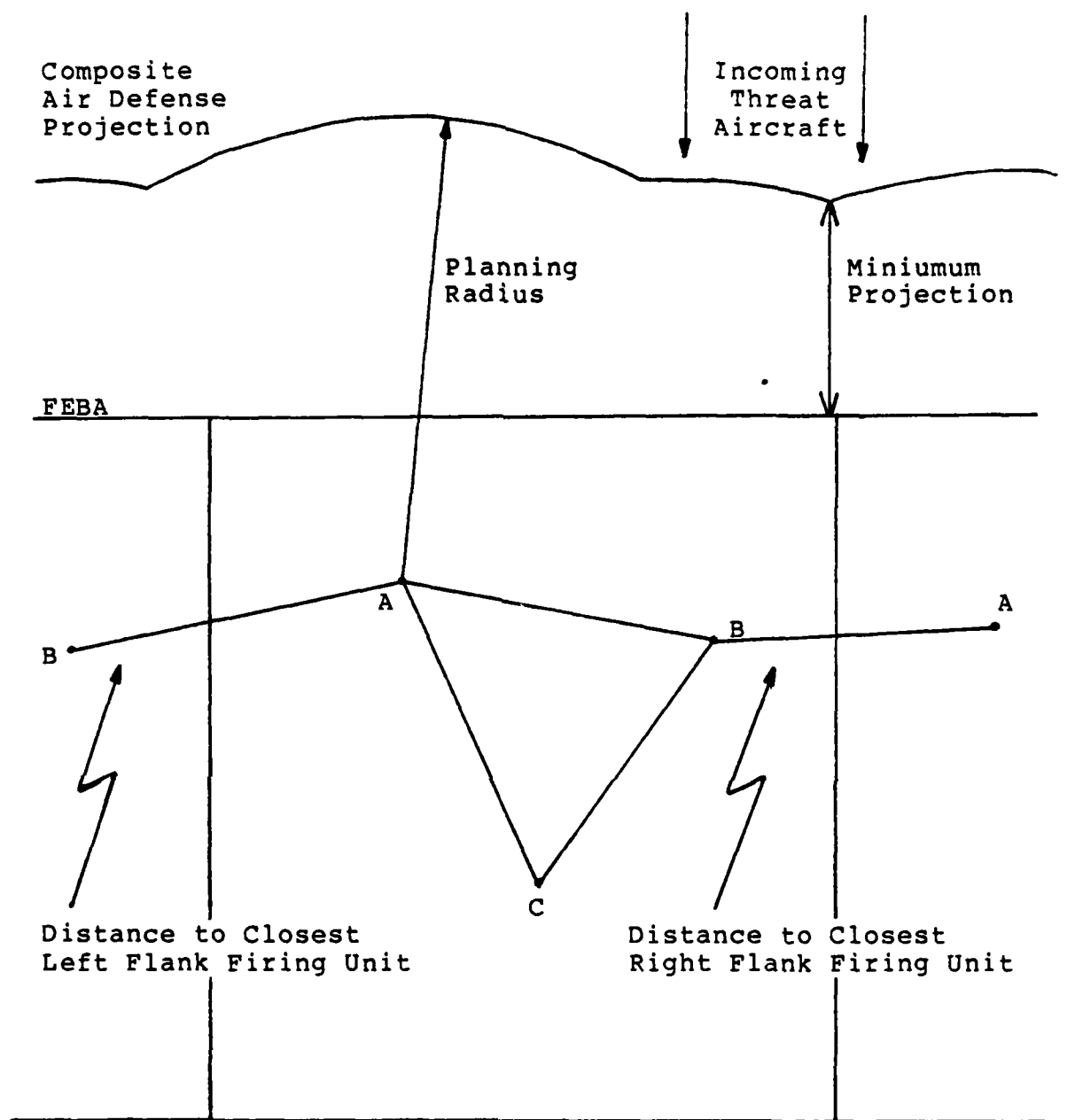
Figure B-1. Mutual Support and Overlapping Fire.

(taken to be four-thirds of the range in SAMPLE). One of the implications of the balance constraint is that "gaps" in the air defense coverage are not allowed; therefore, firing units may not be separated by more than the overlapping fire distance.

Air Defense doctrine also realizes that destroyed firing units do not contribute to division air defense. Therefore, an objective of "reasonable safety" has been established: to the extent that mission objectives are not compromised, firing units will be placed out of range of threat medium field artillery (about 12 kilometers).

Figure 5-1 presented a typical deployment pattern, and is repeated here as Figure B-2. The operating range of the firing units is taken to be 30 km, which implies that mutual support distance is 20 km and overlapping fire distance is 40 km. Balance is achieved through covering practically all of the division area by at least one firing unit. Balance is also achieved across division boundaries by locating the forward firing units within overlapping fire distance of the forward units of adjacent divisions.

The given deployment also satisfies the projection objective to the extent that the air defense coverage is extended past the FEBA. Also, coverage has been weighted



Capital Letters Denote Firing Batteries

Figure B-2. Typical Deployment Scenario

toward the expected avenue of attack (namely, across the FEBA) by placing two batteries in the forward division area. The distances between the firing batteries within the division area, while not quite satisfying the mutual support criterion, are well within overlapping fire distance.

B.3. SITE SELECTION CONSTRAINTS

Doctrinal objectives, weapon system requirements, and available terrain all serve to significantly limit the available firing battery sites. We have chosen to model the site selection task as the constrained choice of weighted site alternatives. Hawk system requirements and the available terrain dictate which points on the division map are possible candidates, while doctrinal objectives provide the means to weigh each candidate. Having weighed each site, the site selection task reduces to choosing the combination of sites with the best composite score.

We have two methods of constraining the site selection process: identifying possible candidates and weighing the chosen candidates. A primary goal of simulator development is to provide user transparency (making the decision process understandable to the real-world expert). Therefore, the identification and

weighing of candidate sites in the simulator should directly relate to the corresponding process performed by the human expert.

In SAMPLE, candidate identification is constrained in two ways. The first is procedural, in that certain portions of the division area are not scanned by the task algorithm; for example, the area within eight kilometers of the FEBA is unsafe to the extent that firing units are not allowed in that area. Also, in the SAMPLE task algorithm, firing units are constrained to lie within the supported division area. The second way identification is constrained is symbolic, in that only "usable" sites are solicited (from the mapreader) as candidates. By making the determination of "usable site" in the mapreader, changes in the definition of usable will not affect the Hawk commander behaviors.

Having determined a possible list of candidates, the problem of weighing the candidates remains. In SAMPLE, this weighing process consists of three steps:

1. For each candidate site, compute a constraint metric for each constraint.
2. Determine the constraint classes based on the values of the metrics.
3. Combine the classes into a score for each candidate.

Constraints. The single-site constraints incorporated into the SAMPLE task algorithm are:

1. Projection.
2. Adjacency -- mutual support between adjacent divisions' firing units.
3. Safety.
4. Accessibility -- nearness to usable roads.
5. Optimality -- balanced placement of firing units.
6. Masking -- dominance of terrain with respect to surrounding area.

Additionally, because the overall deployment plan has three sites, support constraints between firing units are included.

Metrics. Metrics provide a numerical value that can be used to measure the relative "worth" of a particular candidate.

Desired values of projection and adjacency are provided by the user as a part of the scenario definition; the associated metrics simply measure whether these desired values are met by a given site. In both cases, the metric is the ratio of the actual value (based on the candidate's location) to the desired value.

The safety metric is the distance of the candidate from the FEBA, and the accessibility metric is the distance between the candidate and the nearest road accessible by wheeled vehicles.

In the absence of other constraints, the balance objective would yield a symmetric deployment pattern of firing units: the forward firing units would be located a quarter division-width from the division edges, and the rear firing unit would be located on the halfway line. The optimality metric used is the x-coordinate displacement from the balanced configuration.

Emplacement doctrine specifies that the most dominant terrain available is to be used for firing site locations. The masking metric is a simple count of the number of hills that are higher than, and forward of, the candidate site. For firing units within a division, the support metric is the distance between units expressed as a percentage of the planning radius.

Constraint Classes. Once the constraint metrics have been calculated, the value is used to place the candidate into one of four constraint "classes". These classes are used to define candidate sets that are roughly equivalent. Candidates falling into the same class cannot be distinguished as being "better" or "worse"; it is left to other constraints to make the determination. Loosely speaking, the four classes correspond to heuristic definitions of "good" through "poor". In a way, these classes are quite similar to the notion of "fuzzy sets". We believe that this abstraction to four levels

contributes significantly to the transparency of the task algorithm.

Scoring. The final step in the selection process is the scoring of each site in the candidate list. Individual sites are first scored according to the single-site constraint classes just described. Next, forward sites from each side are considered as pairs and the support class of the pair is incorporated. Finally, the pair with the best score is chosen and the result reported.

The constraint classes are assigned an integer value between zero and three with zero representing the best class. The score is computed as a weighted sum of constraint classes.

The six single-site constraints are divided into two groups: projection and adjacency are "mission" constraints (supplied by the user), while the rest are "supplemental" constraints. The class weights were chosen so that changing a mission constraint from one class to the next poorer class is "worse" than the worst-case sum of any three of the other constraints. Since the worst value of any constraint is three, the mission constraint classes are multiplied by a weight of ten before summing to yield the final single-site score. (Three supplemental constraints can change from class "good" to "poor",

resulting in a value change of nine, and still be less severe than a change of one mission constraint class, which would result in a value change of ten.)

The pair score is computed as the sum of the single-site scores of the two candidates, plus twice the support constraint class of the pair.

We believe that this method of calculating pair scores is adequate to model the decision heuristics used by the real-world experts, and simple enough to allow easy understanding of the implementation. Additionally, the weights and class values can easily be adjusted to suit the experiences of the experts for a more accurate representation of the relative importance of the various constraints.

APPENDIX C
SAMPLE USER'S GUIDE

C.1. INTRODUCTION

This appendix is intended to help the SAMPLE user interact with the planning task simulator described in the thesis body. Major topics include how to run the simulator, interfacing with SAMPLE, and the directory structure of the complete simulator system.

Prior to running the system, the user should be familiar with the framework and content of the system structure. As a minimum, the user should read Section 5.5, "User Behavior Specification," Appendix B, "Site Selection Objectives," and Appendix E, "Using Terrain Data in a Planning Simulation."

Because this system is intended to be used as a testbed for further research, user comments are earnestly solicited. Suggestions, criticisms, comments, questions, or identification of bugs should be addressed to the Artificial Intelligence Laboratory, Department of Electrical Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 45433.

C.2. RUNNING SAMPLE

SAMPLE is implemented on the AFIT Scientific Support Computer (a VAX-11/780) running the UNIX operating system.

To run SAMPLE, follow these steps:

1. Log on to the system.
2. In response to the shell prompt,
type: `cd /usr/ai/ROSS/Sample`
3. Type: `sample`
4. In response to the `"->"` prompt,
type: `(load 'continue)`

At this point, SAMPLE is running. After a particular simulation has been completed, the system will prompt the user with: "Do you wish to continue (yes or no)?" An answer of "no" will terminate the session and put the user back in the UNIX shell; "yes" will initiate a new simulation with a different hawk battalion.

C.3. INTERFACING WITH SAMPLE

C.3.1. Input Formatting

The user interacts with SAMPLE in two ways--through defining the simulator scenario, and reading the map for the micro-expert. Before discussing the user interaction, the required input formats will be described.

General. User input is passed to the simulator through the Franz Lisp "reader", which restricts the

possible inputs somewhat. For example, commas have a special meaning to the reader and must be avoided. Also, input is not recognized until a carriage return is typed; this means that mistakes can be corrected using "backspace", but only before the carriage return is typed. The reader incorporates a type-ahead buffer, so that future inputs may be typed in before the present input is processed. Finally, the reader expects input in the form of Lisp "s-expressions." Briefly, an s-expression is either an "atom" or a "list." An atom is either an integer, a floating-point number (which contains a decimal point), or a sequence of characters made up of the letters a-z, the digits 0-9, and the character "-". A list is recursively defined as a sequence of s-expressions enclosed in parentheses. For a further discussion of reader syntax, the user is referred to the Franz Lisp Manual (Foderaro, 1982).

Expected Inputs. For our purposes, most inputs will either be integers or a list of two integers. Locations are entered as a list consisting of the x-coordinate and the y-coordinate; for example, "(-12 -34)<CR>", where the <CR> denotes the carriage return. Note that the list elements are enclosed in parentheses and that commas are not used. Other inputs will either be integers, a "yes<CR>", or a "no<CR>".

Input Diagnostics. During the scenario definition phase, a limited amount of error checking is done. This will allow recovery from certain classes of input errors. However, violation of reader syntax will cause the system to enter an error condition, effectively ending the session. An error condition is identified by an error message followed by a "<l>:" prompt; typing "(exit)<CR>" will return the user to the UNIX shell. No error checking is done during the map-reading phase; therefore, caution is advised when entering map data.

C.3.2. Defining the Scenario

As given in Chapter Five, the scenario definition includes specifying the values of the projection and adjacency constraints, the width of the supported division, and the locations of friendly air defense units.

After a brief introductory message, the system will begin prompting the user for information. The first prompt the user gets is "Enter desired number of hawks in the air defense group (3-7)." The "(3-7)" signifies that the expected response to this prompt is a number between three and seven. Inputs outside of this range are "forced" to the nearest boundary value; for example, if the user inputs "10", then the value used by the system

will be seven (the largest allowable value). Three is the minimum allowed value because the selection algorithm used assumes that the simulated division is flanked on both sides. The upper bound of seven was chosen arbitrarily; values larger than three allow multiple simulations during a single run, but slow the system response time.

The next prompt the user gets is that for the adjacency constraint: "Enter desired adjacency constraint (70-130)." The value of this constraint is the maximum allowed separation between firing units of adjacent divisions, expressed as a percentage of the scenario planning radius. The significance of this constraint, as well as the projection constraint, is covered in Appendix B, "Site Selection Objectives." At this time, it is sufficient to note that lower values of the adjacency constraint produce stronger constraints (i.e., constraints that are harder to meet). Again, if the user attempts to enter a numeric value outside of the given range, the value is set to the nearest extreme.

The third prompt is that for the projection constraint: "Enter desired projection constraint (5-12)." The value of this constraint is the minimum allowed radar projection past the FEBA, expressed in kilometers. In this case, smaller values yield stronger constraints.

The next system output is not a request for input, but a status message: "<creating air defense group>." At this point, the simulator is creating and intializing the dynamic actor instances that make up an air defense group of the specified size. Depending on system load, this may take a while, so the message provides assurance that the system is still running.

The next prompt is for the particular battalion to be simulated. The user is first reminded that only "interior" battalions may be simulated: "Currently, only those battalions with adjacent battalions on each side may be simulated. Which hawk would you like simulated (2-i)?" Here, the "i" stands for one less than the number of hawks in the air defense group.

The division width is prompted for next: "Enter the division width in km (30-60)." The emplacement algorithm used assumes a rectangular division area 60 km deep; therefore, the division width completely specifies the division area. Typical division widths are about 45 km, with the larger widths yielding stronger constraints.

Finally, the locations of friendly air defense units--the Hawk batteries of adjacent divisions and the supported division's Chaperral/Vulcan (C/V) battalion headquarters--are prompted for. All locations are with

respect to a "virtual" coordinate system, with the origin at the intersection of the FEBA and the left edge of the division. The x-axis coincides with the FEBA (increasing x towards the right edge), and the y-axis coincides with the left edge (increasing y away from the division rear). Under this system, the supported division area always lies in the fourth quadrant of a Cartesian coordinate system. At this time, the translation of map coordinates to virtual coordinates is the responsibility of the user. While making the map-reading process somewhat more difficult, the use of virtual coordinates allows for the task algorithm to be developed independently of a particular cartographic system. Also, quite a bit of insight into the decision process can be gained by using "development" terrain (terrain data invented by the designer simply to exercise the system); in this case, the use of other coordinates introduces needless complication.

The values of the friendly locations are unrestricted, but certain values may result in constraints that are impossible to satisfy. Typical values for the siblings' locations are from 8-15 km from the division boundaries, with greater values resulting in stronger constraints that are more difficult to meet. The C/V battalion headquarters is usually located near the center of the division area. While the Hawk battalion headquarters is colocated with the C/V headquarters if

possible, the C/V location does not contribute any additional constraint to the algorithm.

Table C-1 summarizes the components of the scenario definition.

Table C-1. Scenario Definition Components

Component	Minimum allowed	Maximum allowed	Solution difficulty increases with:
number of hawks	3	7	N/A
projection (km)	5	12	increasing values
adjacency (%)	70	130	decreasing values
width (km)	30	60	increasing values
left adjacent Hawk battery	N/A	N/A	decreasing x, decreasing y
right adjacent Hawk battery	N/A	N/A	increasing x, decreasing y
C/V headquarters	N/A	N/A	N/A

C.3.3. Map Reading

The user is responsible for the following map reading functions:

1. Determining how many usable and unusable hills are in a specified region.
2. Determining hill parameters, to include height, location, and the location of the nearest road.
3. Determining the minimum and maximum elevations in a given region.

Definitions.

Hill: for our purposes, a hill is a local maximum in the division area elevations. A hill is usable if

- (a) the local gradient does not exceed 15 degrees,
- (b) it is in "relatively open" terrain (we interpreted this as ruling out cities and dense forest), and
- (c) it contains sufficient room on its forward slope to house a firing platoon (400 by 400 meters).

Elevation: the value of the elevation of the local maximum entered as an integral number of meters.

Hill location: the virtual coordinate pair defining the location of the local maximum. Locations should be entered to the nearest tenth of a kilometer.

Location of nearest road: the location of the nearest road "accessible" from the local maximum. A road is accessible if it does not require crossing an unbridged river or "impassable" symbol on the map.

Minimum elevation: the value of the smallest elevation contour.

Maximum elevation: the value of the largest elevation contour.

C.4. DIRECTORY STRUCTURE

Figure C-1 shows the UNIX file directory structure of the complete simulator system.

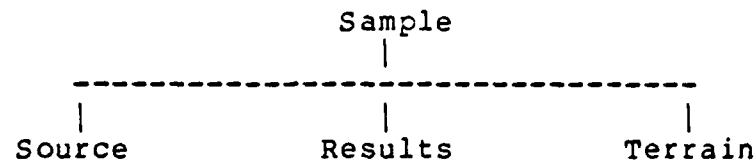


Figure C-1. UNIX Directory Structure.

The Sample directory contains the object files and support files necessary for simulator operation. The Source directory contains the ROSS source code for the various actors, while the Results directory contains several complete scripts of example SAMPLE sessions. The Results directory of the AFIT simulator will also contain a "history" file that documents system changes.

As a prospective user will soon find, the map reading phase of the simulation can become quite tedious, especially if real terrain maps are being used. For this reason, the Terrain directory contains files of "pseudomaps"--lists of hill parameters obtained from an actual 1:50,000 scale map of the East/West German border. The map used covered about 22 by 20 kilometers. Also contained in the Terrain directory are source files containing "helper" functions for organizing the pseudomaps into a form that the user can more readily use during a simulator session.

APPENDIX D
PLANNING TASK SIMULATOR
DESIGNER'S GUIDE

D.1. INTRODUCTION

The purpose of this appendix is to aid future simulator designers in the development of planning task simulators similar to SAMPLE. Prospective designers should be familiar with the ROSS language in general and our thesis goals and assumptions in particular; knowledge of the Lisp programming language is essential for a comprehensive understanding of the system actor and SAMPLE actor code. We therefore suggest that the designer have the ROSS and Lisp manuals available; a copy of the SWIRL (Simulating Warfare in the ROSS Language) manual would also be helpful. SWIRL is an air battle simulator developed at RAND, and it provides specific guidance on simulator development as well as illustrating ROSS programming techniques.

D.2. DESIGN CONVENTIONS

Much of the basic information in this section is drawn from the ROSS and SWIRL manuals; our suggestions are based on our experiences with ROSS and our goal of providing a transparent and flexible example system. Areas covered include reserved symbols, and hints on creating transparent code.

D.2.1. ROSS Reserved Symbols.

The only reserved keywords in ROSS are "ask", "tell" (which are equivalent), and "myself." ROSS is embedded in Lisp, and ROSS commands can be freely intermixed with Lisp function calls. ROSS commands always begin with either ask or tell. "Myself" is a symbol whose value is the actor which is currently executing a behavior.

The characters ! & > + ~ are defined as read character macros in the ROSS read syntax table. ROSS is a non-evaluating language, in contrast to the host language (Franz Lisp). The character "!" signals that the following s-expression (atom or list) be evaluated and the returned value substituted into the command. The character "&" evaluates the following s-expression and "splices" the returned value into the command (that is, enclosing parentheses are removed).

The characters ">" and "+" are pattern-matching symbols. ">" will match any non-nil s-expression, while "+" will match any non-nil sequence of s-expressions. Also, if an atom immediately follows one of these symbols, the value of the atom is set to the matched symbol(s). For example, the atom "Clyde" would match the pattern variable ">person"; furthermore, the value of the symbol "person" would be set to "Clyde."

The character "~" serves as the abbreviate macro; the abbreviation package will be covered in the next section.

D.2.2. Code Transparency

Several features of the ROSS language contribute to the transparency of the developed code. First, Franz Lisp supports symbol names of arbitrary length. This means that names can be chosen that are meaningful to the potential user.

The hyphen character is allowed as part of symbol names, allowing the concatenation of terms into descriptive labels. For example, there is no need to use the name "hcmd" when "hawk-commander" will improve readability.

Second, the pattern matching abilities of ROSS extends this capability to using messages that read much like English. For example, messages such as "choose forward locations using terrain-free rules" can be used, even though the only significant words are "choose", "forward", and "terrain-free" which form the template that the message must match. The use of the abbreviation package further increases the code transparency. This package allows the designer to define abbreviations for awkward phrases and constructs. Without abbreviations, the only way for a leading actor to access its personal memory would be with a message of the form: (ask !(ask !myself recall your memory) fetch <something>). By defining the proper abbreviation, the message becomes (~fetch <something>); obviously, the second form is much more desirable and readable.

Third, Lisp code can be made more readable by the use of the "if" and "loop" macro packages (supplied with the ROSS tape). These packages replace raw Lisp functions such as cond, do, and prog statements. For example,

```
(cond ((eq test t) (do whatever))
      (t (do the other thing)))
```

can be replaced by

```
(if (eq test t)
    then (do whatever)
    else (do the other thing))
```

Similar gains in readability may be obtained by using the loop macro package.

Beyond these macro packages, it is possible to increase the friendliness of Lisp code. For example, the code fragment

```
(if (setq result (caddr lis))
    then (use result)
    else (return))
```

may be somewhat obscure at first. Clearer is

```
(setq result (nthelem 4 lis))
(if (not (null result))
    then (use result)
    else (return nil))
```

The second version does not require the user to figure out that "caddr" references the fourth element of a list, or to remember that a "return" with no argument returns a value of nil. Also, the second version explicitly states that any non-nil value of "result" is sufficient to trigger the "then" clause, and separates computation of "result" from its use in the test clause of the "if" statement.

APPENDIX E
USING TERRAIN IN A
PLANNING SIMULATOR

All battlefield commanders must deal with a common major constraint on their plans: battlefield solutions must be shaped by the terrain. The commander forms his assessments with a map, supplemented with intelligence and reconnaissance data, and sometimes an on-site survey.

A map that is useful for a broad range of commanders (tank, SAM, infantry, etc) requires representation of a variety of data:

1. Height. Height of the land (relative to some benchmark) with sufficient resolution for the commander to discern dangerously sloped or impassable areas.
2. Roads.
3. Waterways.
4. Populated areas.
5. Ground cover. Rubble, sand, swamp, snow (seasonal), etc.
6. Foilage. Brush, forest, etc.

The digital analogue to the commander's map must represent the same generally useful data as the physical map. This general military map representation (GMMR) should contain:

1. Height. Height of the land with sufficient resolution for an automated map-reader to discern dangerously sloped or impassable areas. One representation might be: the mean altitude over 100 meter square areas (tiles).
2. Roads. Roads can be represented as a series of points, a set of vectors or line segments, or simply by marking each tile as passable or impassable. The latter representation however, is only valid in terms of the question "Impassable by whom?", so it is not general.
3. Waterways. Same as roads.
4. Populated areas.
5. Ground cover. Rubble, sand, swamp, snow (seasonal), etc.
6. Foilage. Brush, forest, etc.

The GMMR provides the repository for the information needed by any battlefield commander, real or modelled. However, to be useful in planning a task, it must be viewed in term of the question "What features apply to the task at hand?" The task determines the necessary features that must be abstracted from the GMMR. Each commander utilizes a handful of abstractions plus some detailed knowledge provided by reconnaissance and his on-site survey to reason about the terrain.

Many battlefield planning situations are concerned with the dominance of the terrain. Dominant terrain is conveniently described as "big hills": areas of land that overlook the surrounding region. From the vantage point

of dominant terrain, the commander and his resources (such as radar) have the capability of "seeing" the approach of the enemy at a distance. The dominance provides the commander with a view over other terrain that would mask his view if he had a lower vantage point. To some degree, he can also view areas behind low-lying hills and in valleys.

We have found, through examination of the terrain in Central Europe, that the breadth of a dominant feature is roughly one kilometer. This implies a useful representation of describing terrain using blocks with sides of one kilometer. A block will contain one of four types of terrain:

1. Flat.
2. Low, rolling hills.
3. A dominant mass with multiple peaks.
4. A single dominant hill.

The GMMR we have developed consists of one kilometer square blocks, with each block containing 100 tiles. As described above, a tile is 100 meters on a side. A map of this nature, stored in a data-base, would provide an automated map-reader with the resolution necessary to determine and classify the terrain features. The resolution of the tiles allows the map-reader to determine the gradients of the land. The resolution of the blocks provides the simulated commander with an abstraction of the terrain in terms of dominance.

The abstraction can be used to describe two-dimensionally the areas visible from the site and those areas masked by other terrain. The third dimension, elevation with respect to the site, is represented by the following heuristics:

1. Masking decreases as the elevation angle increases, for the simple reason that gross terrain features get smaller at the top. The inverse is also true, masking increases with decreasing elevation angles.
2. Look-down capability is enhanced with larger (negative) elevation angles. This is the ability to see behind features due to the viewer's higher vantage point.

Populated areas, ground cover, and foilage can be represented at the same or lower resolution than height information, due to the dynamics of encroachment. The edges of the region are fluid, and boundary conditions around the region are not much different than conditions within, in terms of mobility of the commander's assets. A low resolution representation will suffice. Additionally, the commander does not make detailed judgements of cover (for camouflage and shielding) from his map. Rather, he notes the fact that the areas of cover exist, and makes the specific decisions opportunistically during the occupation or the battle.

VITAS

Gregg Harold Gunsch was born to Harold and Delores Gunsch on 30 May 1958 in Eureka, South Dakota. He graduated from high school in Bismarck, North Dakota in 1976. He then attended Bismarck Junior College for one year and the University of North Dakota for two, where he received the degree of Bachelor of Electrical Engineering in May 1979. Upon graduation, he received a commission in the USAF through Officer's Training School. After technical training, he was assigned to the 394th ICBM Test Maintenance Squadron, Vandenberg AFB, California in support of the Minuteman Operational Test and Evaluation Program. There he served as a Technical Engineering Team Chief, and then OIC, ICBM Engineering, until entering the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: 1722 North 21st St.

Bismarck, North Dakota 58501

Bob Victor Hebert was born to Iris and Robert Hebert on 12 January 1957 in Houma, Louisiana. He graduated from high school in 1973 and attended Nicholls State University for three years. He enlisted in the USAF in January 1977 and completed technical training at Keesler AFB, Mississippi. He served as an avionics communications specialist in the 1st Special Operations Wing, Hurlburt Field, Florida until September 1980. He attended Georgia Institute of Technology under the Airman's Education and Commissioning Program and received the degree of Bachelor of Electrical Engineering in September 1980. Upon graduation, he received a commission in the USAF through Officer's Training School. He then served as a command and control analyst in the Foreign Technology Division, Wright-Patterson AFB, Ohio until entering the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: PO Box 456

Napoleonville, Louisiana 70390

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/EE/83D-24		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Foreign Technology Div	8b. OFFICE SYMBOL (If applicable) FTD/TQCS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code) Wright-Patterson AFB, Ohio 45433		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See Box 19		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT NO.
12. PERSONAL AUTHOR(S) Gregg H. Gansch, B.S., Captain, USAF Bob U. Hebert, B.S., 1Lt, USAF			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1983 December	15. PAGE COUNT 135
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	SUB. GR.	
09	02	EXPERT SYSTEMS MILITARY PLANNING ROSS ARTIFICIAL INTELLIGENCE SIMULATORS	

Approved for public release: LAW AFR 180-17.

Lynn E. Wolaver
LYNN E. WOLAVER

Dean for Research and Professional Development

Air Force Institute of Technology (AFIT)
Wright-Patterson AFB, Ohio 45433

7 Feb 84

Box 19 continued:

Portions of the simulator have been implemented as a demonstration of the feasibility of this approach, as well as a guide for future developers. The language used was ROSS, an object-oriented simulation language embedded in Franz LISP or MacLisp and developed by the Rand Corporation. The implementation was done on the Air Force Institute of Technology's Scientific Support Computer, a VAX 11/780. The dialect used was the Franz LISP version of ROSS.

Several system functions such as data storage and retrieval mechanisms, a communications channel model, design aids, and debugging aids were implemented to provide a workbench for the development of an extended simulator. One task specialist was also constructed: a small-scale expert system that performs the planning involved in selecting emplacement sites for Hawk medium-range surface-to-air missile (SAM) batteries.

Although the implemented task specialist is highly constrained and only an approximation of the real-world counterpart, it serves to demonstrate the ease with which task specialists can be constructed. The transparency, modularity, and power of ROSS, with the additional system functions, provides a robust environment in which to simulate real-world planning algorithms.

